Санкт-Петербургский государственный университет

ВОРОШИЛОВ Александр Алексеевич

Выпускная квалификационная работа

Использование методов искусственного интеллекта в технологиях распределенных реестров

Уровень образования: магистратура
Направление: 02.04.02 «Фундаментальная информатика и информационные технологии»
Основная образовательная программа: ВМ.5502 «Вычислительные технологии»

Научный руководитель:
доцент кафедры КММС,
кандидат физ.-мат. наук,
Корхов В.В.

Рецензент: Директор
ООО «ОСЕНСУС АРМ»,
кандидат наук СПбГУ,
Абраамян С.А.

Санкт-Петербург
2020 год

Оглавление

Введение

С развитием вычислительных мощностей стало возможно совершенствование систем искусственного интеллекта. Сейчас искусственный интеллект внедрен в огромное число различных областей. Система искусственного интеллекта – это довольно абстрактное понятие, под которым обычно понимают систему, способную воспринимать окружающую среду и предпринимать действия, которые максимизируют шансы системы на успешное достижение поставленных целей. Искусственный интеллект часто используется в сложно формализуемых задачах благодаря возможности автоматического выделения особенностей исходных данных и способности к обобщению. Также внедрение систем искусственного интеллекта может значительно сократить расходы на высококвалифицированных экспертов в различных областях и исключить ошибки, связанные с человеческим фактором.

Появление распределенных реестров сильно повлияло на развитие IT. Технологии распределенных реестров (Distributed ledger technology, DLT) можно рассматривать как технологии хранения данных, в которых отсутствует доверие между узлами. Ключевыми особенностями распределенных реестров являются:

- отсутствие центрального администратора;
- предоставление возможности совместного использования;
- обеспечение синхронизации между несколькими узлами (которые могут принадлежать различным государствам или учреждениям).

Распределенный реестр представляет из себя одноранговую (пиринговую) сеть, в которой информация хранится сразу на нескольких узлах. Между узлами сети отсутствует доверие, поэтому для обновления данных реестра участники сети должны прийти к консенсусу – некому заранее установленному условию разрешения конфликтов.

Существует множество различных технологий распределенных реестров:

- Blockchain [1],
- Hashgraph [2],
- Направленный ациклический граф (Directed Acyclic Graph, DAG) [3], и т.д.

Среди всех распределенных реестров, наибольшую популярность получила технология блокчейн, что связано с появлением криптовалюты Bitcoin [4].

Взаимодействие систем искусственного интеллекта с различными DLT является довольно перспективной областью исследований, так как это дает возможность использовать преимущества искусственного интеллекта в распределенных системах, в которых отсутствует доверие между узлами. Также интеллектуальные системы могут быть применены для анализа и оптимизации распределенных реестров.

Постановка задачи

Сеть блокчейн является анонимной, все операции происходят только между адресами пользователей. Многие блокчейн-сети (например, Bitcoin, Ethereum [5]) являются публичными. Это означает, что стать участником сети может каждый. Кроме того, имеется открытый доступ ко всей цепочке транзакций.

Несмотря на публичность транзакций, лица или организации, стоящие за адресами сети, остаются неизвестными. Также остается неизвестной роль пользователей в сети. Однако, по информации о транзакциях можно классифицировать пользователей на различные типы. Это может быть интересно для понимания вектора развития сети. Также, выделив особенности поведения различных групп пользователей, можно определить, к какой группе относится конкретный адрес.

В данной работе была поставлена задача классификации адресов по заранее известным типам. Также необходимо провести сравнение различных методов классификации с целью выбора оптимального.

При получении набора размеченных по группам адресов и списков транзакций для них, появляется возможность провести обучение с учителем. Таким образом, используя методы искусственного интеллекта, можно будет определить роль конкретного адреса в сети.

Для исследования была выбрана децентрализованная платформа Ethereum на базе распределенного реестра Blockchain.

Обзор литературы

Блокчейн был разработан человеком (или группой людей), под псевдонимом Сатоши Накамото (Satoshi Nakamoto) в 2008 году в качестве основы для криптовалюты Bitcoin [4].

Блокчейн представляет из себя связанный список, элементами которого являются блоки транзакций (рис. 1). Каждый блок содержит список транзакций, хэш предыдущего блока и число nonce, за счет которого обеспечивается механизм консенсуса Proof of Work, который также был введен в данной работе.

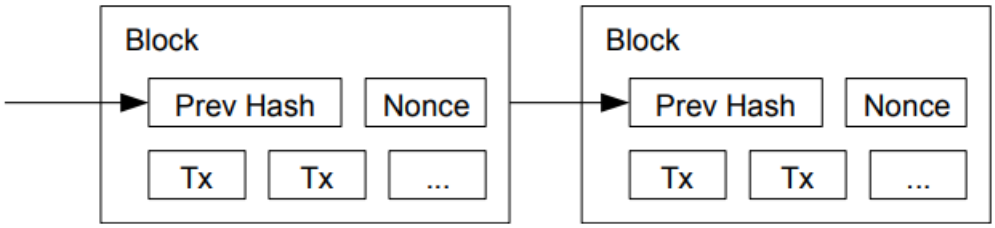


Рис. 1. Схема цепочки блоков.

Задачу формирования и валидации блоков выполняют майнеры. Суть майнинга заключается в подборе такого числа nonce, чтобы хэш блока имел в начале некоторое количество нулевых битов. Количество необходимых битов определяет сложность майнинга. Т.к. не существует алгоритма определения числа nonce, майнинг сводится к последовательному перебору этого числа, что требует серьезных вычислительных мощностей.

Таким образом обеспечивается безопасность (на основе консенсуса Proof of Work), поскольку для подмены транзакции в блоке становится необходимым пересчитать хэши всех предыдущих блоков, что означает новый подбор всех чисел nonce. Также Proof of Work подразумевает, что злоумышленник не станет компрометировать систему, активным участником которой он является.

Алгоритм Proof of Work имеет некоторые недостатки, такие как:

- процесс принятия нового блока в цепочку довольно медленный;
- вычислительные мощности расходуются впустую;
- подвержен атаке 51% (если участник сети имеет 51% вычислительных мощностей, то он может контролировать всю сеть).

По этим причинам разработано множество других алгоритмов консенсуса, однако, несмотря на все недостатки, алгоритм показывает хорошую работоспособность и устойчивость.

Смарт-контракт является компьютерным протоколом, предназначенным для заключения договоров и исполнения контрактов в цифровом виде. Это позволяет мгновенно совершать транзакции без участия третьих лиц.

Идея создания смарт контрактов появилась еще в 1990-е годы, термин ввел ученый в области информатики, криптографии и права Ник Сабо (Nick Szabo) в статье [6]. Однако реальное их применение стало возможным лишь с появлением технологии blockchain.

Возможность создания полноценных смарт-контрактов на полном по Тьюрингу языке программирования (Solidity) впервые появилась в сети Ethereum [7], с чем связана ее популярность. Эта особенность делает Ethereum не просто криптовалютой, а площадкой для создания децентрализованных приложений [5]. Единицей валюты сети Ethereum является эфир (ETH).

Благодаря использованию смарт-контрактов в блокчейн системе, становится невозможным переписать программный код контракта или воспрепятствовать его выполнению. Иначе говоря, сделки на основе смарт-контрактов перманентны и не могут быть изменены после заключения.

Несмотря на все преимущества смарт-контрактов, они имеют ряд недостатков, связанных со сложностью интеграции с реальным миром и юридическими вопросами.

На данный момент существует множество работ, описывающих различные методы интеграции и взаимодействия искусственного интеллекта с распределенными реестрами. В статье “Decentralized & Collaborative AI on Blockchain” [8] Джастин Харрис (Justin D. Harris) и Бо Вагонер (Bo Waggoner) описывают платформу для обмена и улучшения модели машинного обучения. Благодаря данной платформе пользователи могут свободно получить доступ к прогнозам модели или предоставить данные, которые помогут улучшить модель.

Также работе с моделями машинного обучения посвящена статья Бесира Куртулмуса (Besir Kurtulmus) и Кенни Даниэла (Kenny Daniel) “Trustless Machine Learning Contracts; Evaluating and Exchanging Machine Learning Models on the Ethereum Blockchain” [9]. Авторы предлагают новый протокол поверх блокчейна Ethereum, который создает рыночную площадку для обмена моделями машинного обучения в автоматическом и анонимном режиме для участников.

Большое число научных работ посвящено исследованию распределенных реестров методами искусственного интеллекта. Аджай Сингх (Ajay Singh) в своей диссертации “Anomaly detection in the Ethereum network” [10] сравнивает различные методы искусственного интеллекта для определения аномального поведения пользователей в сети Ethereum по их транзакциям.

В работе “Machine Learning in/for Blockchain: Future and Challenges” [11] Фанг Чен (Fang Chen), Хонг Ван (Hong Wan), Хуа Цай (Hua Cai) и Гуан Чэн (Guang Cheng) рассматривают несколько различных вариантов совместного использования блокчейна и искусственного интеллекта:

1. Классификация транзакций блокчейна;

2. Прогнозирование стоимости криптовалюты Bitcoin;
3. Федеративное обучение моделей [12]. Мобильные устройства создают транзакции, и, если большинство узлов подтверждает, что производительность модели стала выше, то обновления внедряются в модель;
4. Распределение нагрузки для мобильного майнинга.

В статье “Classifying Ethereum users using blockchain data” [13] Матиас Де Алиага (Matthias De Aliaga) исследует различные архетипы пользователей сети Ethereum и выделяет 5 основных категорий:

1. Exchange (цифровые торговые площадки, где трейдеры могут покупать и продавать криптовалюту);
2. Mining Pools (майнинговые агрегаторы, которые позволяют объединять вычислительную мощность нескольких майнеров, после чего распределяют между ними вознаграждение);
3. Token Wallets (Кошельки, которые хранят ETH, собранный в результате ICO);
4. Investors (крупные держатели ETH, собранного на раннем этапе ICO, часто проявляющие активность на рынке);
5. Whales (“Киты” – держатели большой суммы ETH, которые при желании могут значительно влиять на рынок).

Статья Уилла Прайса (Will Price) “Clustering Ethereum Addresses” [14] посвящена решению задачи кластеризации пользователей сети Ethereum по их транзакциям. В работе были приведены метрики, выделенные из списка транзакций, на основе которых проводилась кластеризация. В результате, для различных групп пользователей были построены диаграммы, демонстрирующие степени влияния выделенных метрик.

Глава 1. Обзор и сравнение классификаторов

1.1. Обзор методов классификации

Выбор метода классификации сильно зависит от поставленной задачи и собранных данных. Не существует единого алгоритма выбора наилучшего решения, выбор производится исходя из опыта и с опором на статьи по смежной тематике.

Задача классификации адресов сети Ethereum мало исследована, о чем говорит дефицит статей по данной тематике. Существующие статьи, как правило, не содержат приемлемого описания технологии классификации и по большей части носят публицистический характер.

Данные обстоятельства не позволяют однозначно определиться с методом классификации, поэтому одной из целей данной работы является анализ методов для выбора наилучшего решения поставленной задачи.

Существуют некоторые рекомендации по выбору классификаторов от компаний, занимающихся анализом данных и библиотек по машинному обучению [15, 16, 17]. Опираясь на эти рекомендации, для исследования были выделены следующие методы:

- Метод k-ближайших соседей
- Метод опорных векторов
- Решающие деревья
- Случайный лес
- Градиентный бустинг
- Искусственные нейронные сети

1.1.1. Метод k-ближайших соседей

Метод k-ближайших соседей (k-nearest neighbors, KNN) [18] основан на оценке метрики расстояния между точками в n-мерном пространстве. Основным параметром метода является k – количество соседей. При классификации алгоритм определяет k «ближайших соседей» объекта и относит его к классу большинства. На рисунке 2 представлен пример бинарной классификации в двумерном пространстве с параметром $k = 5$.

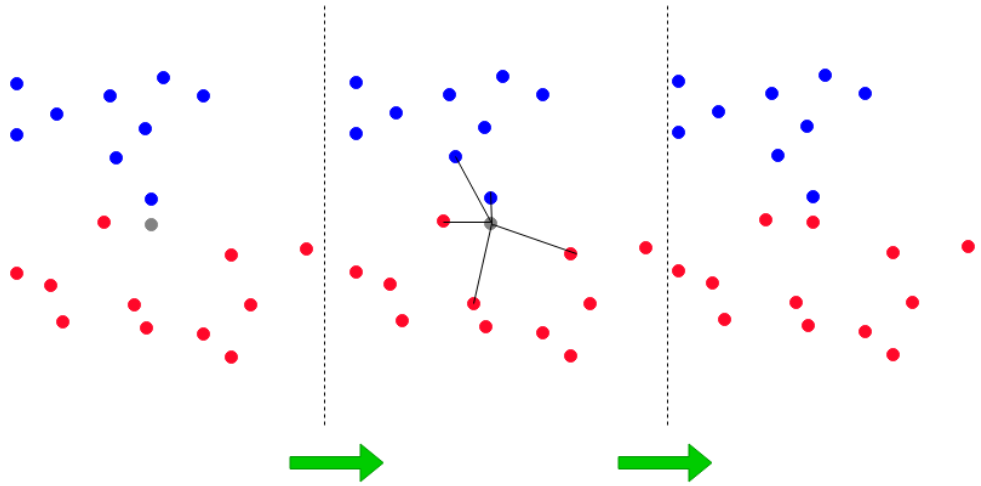


Рис. 2. Пример работы метода k-ближайших соседей.

Метод является довольно простым в реализации, но несмотря на это в некоторых задачах показывает высокую эффективность.

1.1.2. Метод опорных векторов

Суть метода опорных векторов (support vector machine, SVM) [19] заключается в разделении элементов разных классов гиперплоскостями в n-мерном пространстве. Ниже представлена схема работы алгоритма (рис. 3).

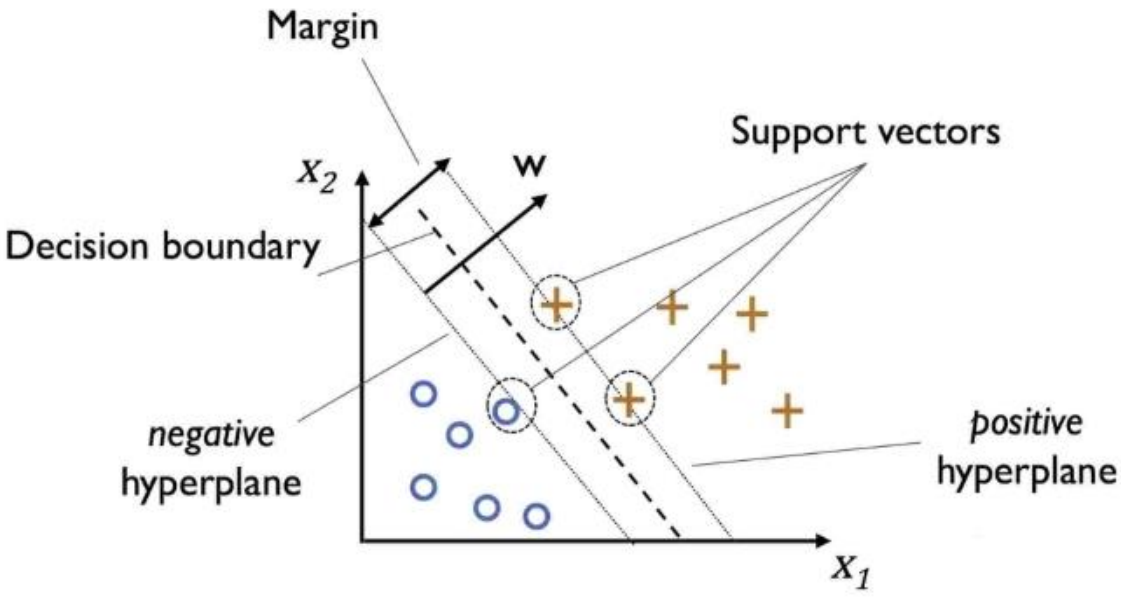


Рис. 3. Схема работы метода опорных векторов.

Выборка, которую удается разделить на классы гиперплоскостями, называется линейно разделимой. В таком случае гиперплоскости выбираются таким образом, чтобы между ними не лежало ни одной точки. После этого расстояние между гиперплоскостями максимизируется, а решающая гиперплоскость выбирается на равном удалении от них. Делается это из предположения о том, что ошибка классификации уменьшается при увеличении расстояния между точками и решающей плоскостью.

На практике редко встречаются линейно разделимые выборки, часто возникают выбросы, которые размывают границы между классами. В таком случае условия разделения смягчаются, вводится параметр ξ, который характеризует величину ошибки классификации (рис. 4). В последствии необходимо добиться того, чтобы параметры были минимальны.

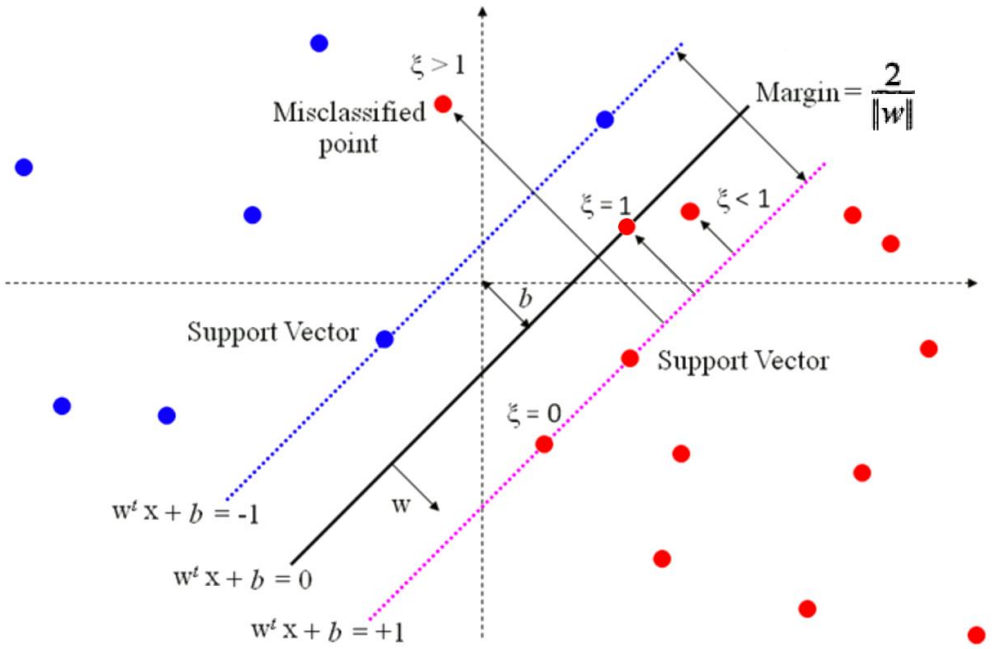


Рис. 4. Схема работы метода опорных векторов с параметром ξ.

Иногда удается решить проблему линейной неразделимости предполагая, что существует пространство большей размерности, в котором выборка становится линейно разделимой.

Так как весь процесс построения гиперплоскостей сводится к работе со скалярными произведениями векторов, повышение размерности осуществляется за счет замены скалярного произведения на так называемое ядро – функцию, являющуюся скалярным произведением в новом пространстве с линейно разделимой выборкой (рис. 5).

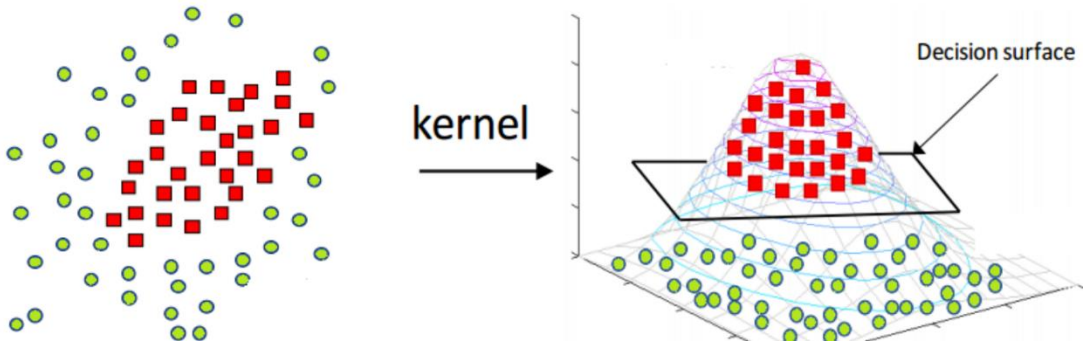


Рис. 5. Пример использования ядра в методе опорных векторов.

Метод опорных векторов является хорошо изученным и отлично справляется с рядом задач, однако в некоторых случаях применение этого метода оказывается затруднительным. Главной причиной является слишком сильное влияние выбросов на результат классификации. Также нужно отметить, что не существует метода наиболее оптимального подбора ядра для классификации линейно неразделимой выборки.

1.1.3. Решающие деревья

Решающие деревья (Decision Tree) [20] являются инструментом поддержки принятия решений, основанном на древовидной модели, содержащей условные операторы. В узлах дерева содержатся условия на входные данные, а в листьях – значения целевой функции. На рисунке 6 представлен пример дерева решений для системы кредитования.

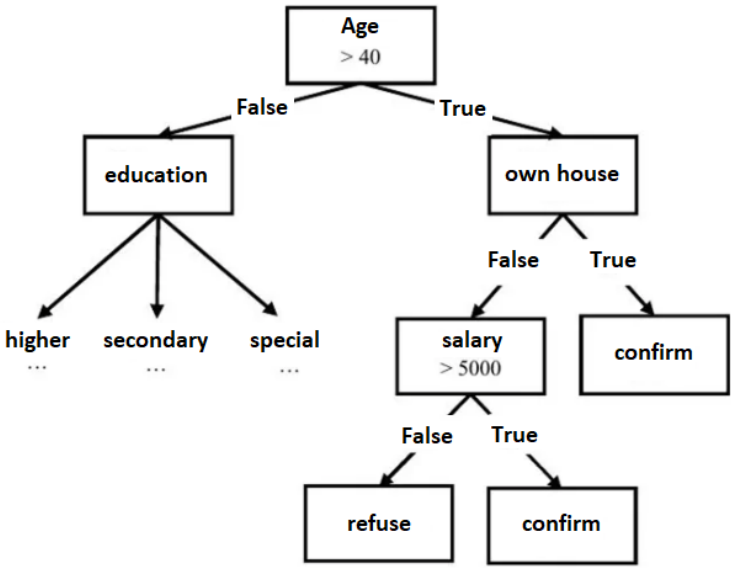


Рис. 6. Пример дерева решений для системы кредитования.

Для описания процесса обучения решающих деревьев используется понятие энтропии. Пусть система имеет i состояний, вероятности нахождения в которых равны p_i. Тогда энтропия Шеннона имеет вид:

$$S = -\sum_{i}^{N} p_i \log_2 p_i$$

Энтропия характеризует степень хаоса в системе, т.е. чем она меньше, тем более упорядоченной является система.

Существуют различные методы построения деревьев решений, но в самых популярных из них (ID3 [21], C4.5 [22]) лежит принцип жадной максимизации прироста информации. Его суть заключается в следующем: на каждом шаге алгоритм выбирает признак обучающего множества таким образом, чтобы разбиение на нем привело к наибольшему приросту информации. Процедура разбиения выполняется рекурсивно до тех пор, пока энтропия не станет достаточно мала.

В процессе построения дерева можно добиться нулевой энтропии, но это приведет к переобучению модели (overfitting) – явлению, при котором модель показывает хорошую точность на обучающем множестве, но плохо работает на реальных данных. Можно сказать, что при переобучении модель просто подстраивается под обучающее множество.

Помимо энтропии существуют и другие критерии качества разбиения. Одним из наиболее популярных является неопределенность Джини (Gini impurity). На практике использование неопределенности Джини дает схожие результаты с использованием понятия энтропии.

Решающие деревья являются очень популярным методом классификации, часто лежащим в основе других более мощных методов. Метод прост в интерпретации и не требует подготовки данных (например, для других методов часто приходится проводить нормализацию) и быстро работает на любых массивах данных. Также для метода существуют алгоритмы оценки важности признаков (feature importance), которые позволяют оптимизировать работу

модели под конкретную задачу путем исключения слабо влияющих на результат работы признаков.

Одной из главных проблем метода является переобучение, бороться с которым можно регулируя глубину дерева. Однако, невозможно заранее точно определить максимально допустимую глубину, этот параметр, как правило, подбирается эмпирически.

1.1.4. Случайный лес

Случайный лес (Random Forest) [23] – алгоритм, идея которого заключается в использовании набора (ансамбля) решающих деревьев. Построение модели осуществляется с использованием подхода feature bagging:

1. Деревья строятся на случайной подвыборке обучающего множества;
2. При формировании узлов деревьев выбираются случайные параметры (feature bagging).

Первая особенность означает, что каждое дерево строится на основе своего собственного обучающего множества, формирующегося из основного выборкой с возвращением или бутсраппингом (bootsrapping). Из выборки размерности N формируется новая выборка той же размерности, каждый элемент которой берется как случайный элемент базовой выборки. Таким образом есть вероятность, что элемент войдет в выборку несколько раз или не войдет в нее вовсе. Это позволяет создавать на одной выборке большое число относительно некоррелированных деревьев.

Вторая особенность также направлена на увеличение многообразия деревьев. При построении классической модели дерева решений, каждый узел выбирает признак, наилучшим образом разделяющий данные. Случайный лес

наоборот ограничивает количество признаков некоторым случайным подмножеством.

Все эти особенности направлены на повышение разнообразия и снижение корреляции между моделями. Итоговое решение принимается большинством деревьев. Схема работы алгоритма представлена ниже (рис. 7).

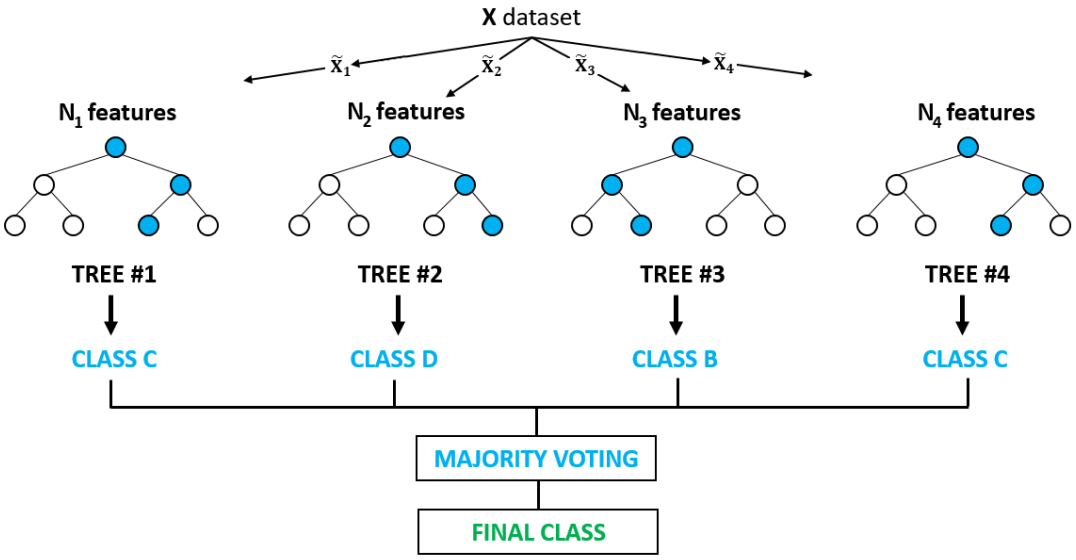


Рис. 7. Схема работы случайного леса.

Большим преимуществом данной модели является способность эффективно работать с данными, имеющими большое число признаков. Также метод не требует предобработки данных (так как в основе лежат решающие деревья), хорошо параллелизуется и масштабируется.

Из недостатков можно отметить большой размер конечной модели, однако, на практике, как правило, это не представляет серьезной проблемы.

1.1.5. Градиентный бустинг

Идея градиентного бустинга (Gradient Boosting) [24] заключается в использовании ансамбля более слабых моделей. Обычно, в качестве этих

моделей выступают решающие деревья. Процесс обучения является итеративным, в ходе которого аддитивная модель расширяется. На каждой итерации в ансамбль добавляется новая модель, которая должна увеличивать точность классификации. Подбор параметров производится путем минимизации функции ошибки градиентным спуском. На рисунках 8 и 9 схематично изображен принцип работы классификатора на решающих деревьях.

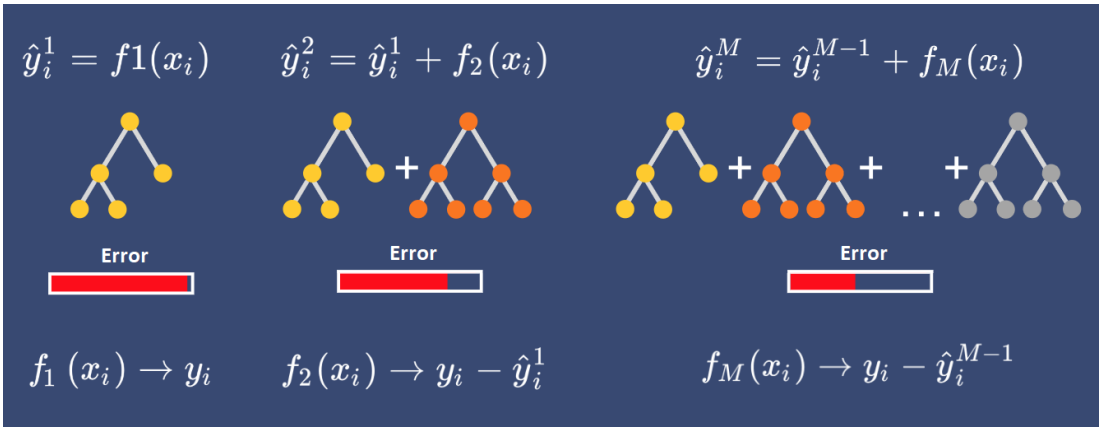


Рис. 8. Схема работы градиентного бустинга.

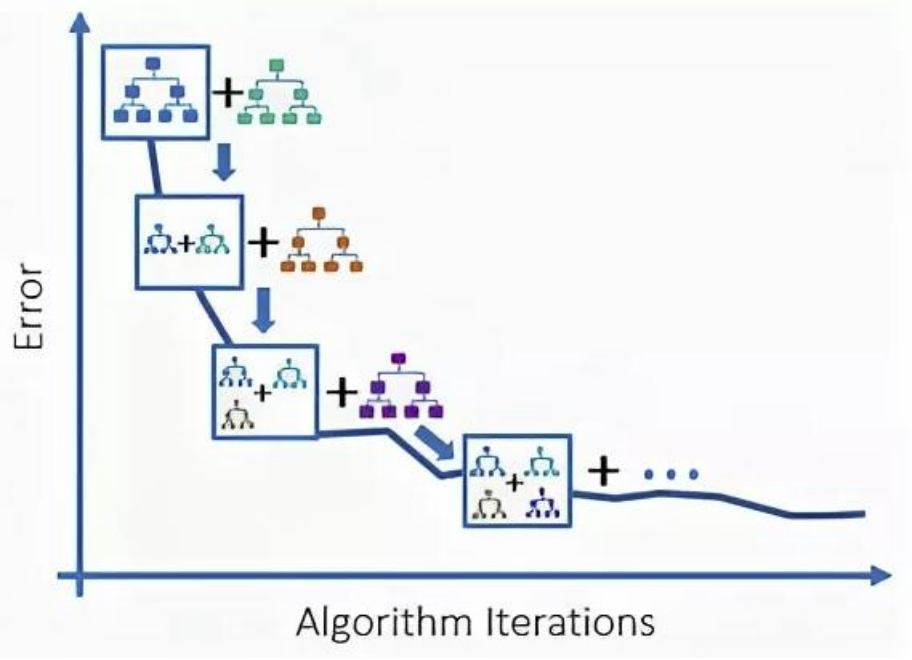


Рис. 9. График уменьшения ошибки классификации с увеличением размера модели градиентного бустинга.

Градиентный бустинг является одним из самых эффективных алгоритмов, благодаря аддитивной природе формирования ансамбля моделей. Также метод довольно гибок, что позволяет настроить модель наилучшим способом. Метод позволяет самостоятельно выбирать функцию потерь и формировать ансамбли из различных комбинаций базовых моделей, благодаря чему можно тонко настроить классификатор под конкретную задачу.

Одним из недостатков метода является медленный процесс построения модели. Это связано с большим количеством параметров, которые необходимо оптимизировать на каждой итерации. Также метод плохо работает с мощными базовыми моделями. Их использование сильно увеличивает сложность формирования модели, при этом, как правило, существенного прироста в точности классификации не происходит.

1.1.6. Искусственные нейронные сети

Искусственная нейронная сеть (Artificial Neural Network) [25] – математическая модель, построенная по аналогии с биологическими нейронными сетями. Ключевым элементом искусственных нейронных сетей является искусственный нейрон – упрощенная математическая модель естественного нейрона [26]. Нейрон имеет несколько входов и один выход (рис. 10). На входы подается вектор значений, нейрон вычисляет выход и отправляет его на входы следующим нейронам.

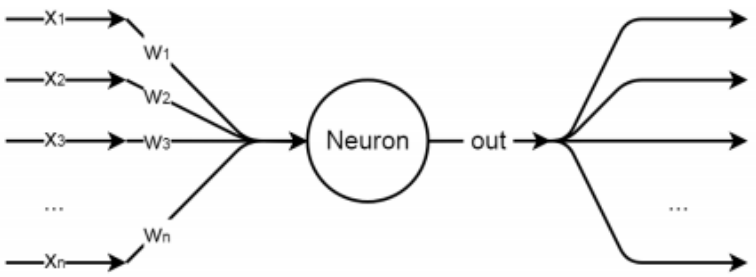
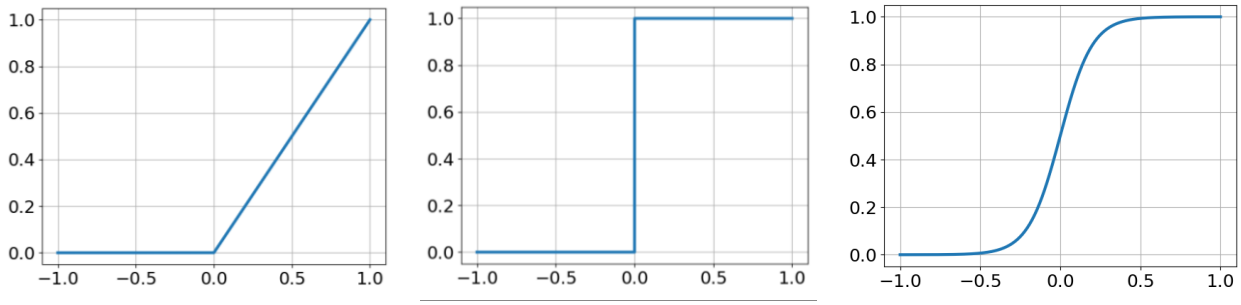


Рис. 10. Схема искусственного нейрона.

На каждом входе x_i нейрон имеет весовые коэффициенты w_i. Также имеется дополнительный скрытый вход x_0 со своим весом w_0. Для вычисления выхода нейрона необходимо получить взвешенную сумму входов и вычислить значение функции активации от этой суммы:

$$s = \sum_{i=0}^{n} w_i x_i \ , \qquad out = f(s) \ .$$

Скрытый вход x_0 отвечает за смещение функции активации вдоль оси OX. Как правило на практике применяются следующие виды функции активации:



a) Полулинейная функция b) Пороговая функция c) Сигмоидальная функция

Рис. 11. Пример функций активации.

Нейронная сеть состоит из большого числа нейронов, связанных друг с другом. Для решения задачи классификации разработано большое число различных архитектур нейронных сетей. Для классификации данных, являющихся количественными признаками каких-либо объектов или явлений, как правило используется многослойный персептрон (multilayer perceptron, MLP) [27].

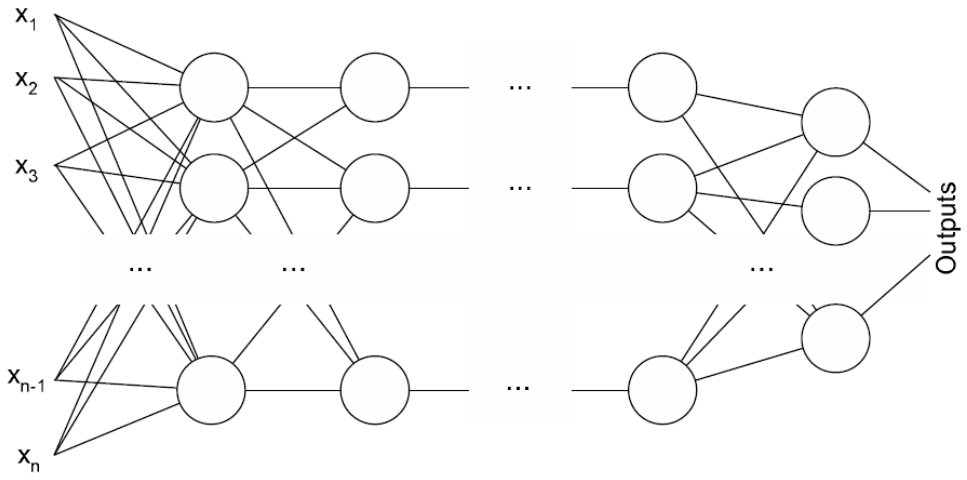


Рис. 12. Схема многослойного персептрона.

Многослойный персептрон состоит из нескольких слоев нейронов (рис. 12). Нейроны между слоями связываются согласно таблице связей, но на практике принято формировать связи по схеме «каждый с каждым», так как в процессе обучения лишние связи будут ослабляться.

Данная архитектура достаточна универсальна, с чем связана ее популярность. Персептрон широко применяется для решения задач аппроксимации, классификации и прогнозирования. Более того, многослойный персептрон решает любую задачу интерполяционной постановки. Главными недостатками можно считать большое количество эмпирически подбираемых параметров обучения и существенную зависимость архитектуры от постановки задачи.

Обучение модели происходит методом обратного распространения ошибки (backpropagation) [28]. Суть метода заключается в распространении сигналов ошибки от выхода сети к ее входу, изменяя при этом веса нейронов таким образом, чтобы минимизировать ошибку. Сеть обучается на каждом элементе обучающего множества несколько раз. Цикл обучения, в процессе которого сети были предоставлены все элементы обучающего множества, называется эпохой обучения. Как правило, количество эпох обучения устанавливается заранее, и

должно подбираться таким образом, чтобы свести к минимуму ошибку классификации на тестовом множестве. Процесс обучения может занимать много времени, так как необходимо оптимизировать большое количество параметров.

Одной из главных проблем обучения является так называемый overfitting или переобучение. Для этого следует выбирать такое количество эпох обучения, при котором точность на тестовом и валидационном множествах перестает расти. Также для решения этой проблемы был разработан метод регуляризации Dropout [29]. Суть метода заключается в исключении некоторого процента случайных нейронов из обучения на каждой эпохе или итерации. Данный метод позволяет увеличить скорость обучения, а также повышает обобщающую способность модели.

Также для ускорения процесса обучения часто применяется пакетная нормализация (Batch Normalization) [30]. В некоторых случаях данная технология позволяет повысить точность классификации. Подход основан на нормализации данных, передаваемых в функцию активации нейронов.

При проектировании архитектуры нейронной сети следует руководствоваться статьями по схожей тематике, различными рекомендациями и отталкиваться от собственных экспериментов.

Таким образом, нейронные сети позволяют решать большое число различных задач с высокой точностью, однако существует ряд плохо изученных вопросов, решение которых возможно лишь эмпирическим путем.

1.2. Метрики оценки классификации

Для анализа работы классификаторов как правило используют следующие метрики:

- Accuracy,
- Precision,
- Recall,
- F1-score.

Для описания данных метрик вводят матрицу ошибок (confusion matrix) [31]. В задаче бинарной классификации матрица имеет следующий вид:

Таблица 1.

Матрица ошибок в задаче бинарной классификации.

Результат классификации	Реальная принадлежность к классу	
	$y = 1$	$y = 0$
$\tilde{y} = 1$	True Positive (TP)	False Positive (FP)
$\tilde{y} = 0$	False Negative (FN)	True Negative (TN)

Матрица ошибок разбивает результаты работы модели на 4 группы:

- True Positive (TP) – модель верно предсказала принадлежность элемента к классу;
- False Positive (FP) – модель предсказала принадлежность элемента к классу, однако объект классу не принадлежит (ошибка 1-го рода);
- False Negative (FN) – модель ошибочно показала, что элемент не принадлежит классу (ошибка 2-го рода);

- True Negative (TN) – модель верно показала, что элемент не принадлежит к классу.

Для задачи небинарной классификации матрица будет иметь немного другой вид. Рассмотрим вид матрицы ошибок для нашей задачи относительно класса “Exchange”:

Таблица 2.

Матрица ошибок для класса “Exchange”.

<table>
<tr><td rowspan="2">Результат классификации</td><td colspan="4">Реальная принадлежность к классу</td></tr>
<tr><td>Exchange</td><td>ICO Wallets</td><td>Mining</td><td>Token Contract</td></tr>
<tr><td>Exchange</td><td>TP</td><td colspan="3">FP</td></tr>
<tr><td>ICO Wallets</td><td rowspan="3">FN</td><td colspan="3" rowspan="3">TN</td></tr>
<tr><td>Mining</td></tr>
<tr><td>Token Contract</td></tr>
</table>

Теперь рассмотрим метрики.

1.2.1. Accuracy

Accuracy является самой простой метрикой для оценки точности. Это доля правильных ответов модели:

$$Accuracy = \frac{RIGHT}{ALL}.$$

Данная метрика не в полной мере оценивает работу алгоритма на неравномерно распределенных данных.

1.2.2. Precision и recall

Часто требуется оценить работу классификатора на конкретном классе. Precision (точность) и recall (полнота) хорошо подходит для данной оценки.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Precision показывает, какая доля объектов, отнесенных классификатором к определенному классу, является верной. Recall в свою очередь показывает, какая доля объектов класса была правильно классифицирована.

1.2.3. F1-мера

Для того, чтобы агрегировать precision и recall в одну метрику, используют F-меру:

$$F_\beta = (1 + \beta^2)\frac{precision \cdot recall}{\beta^2 \cdot precision + recall}.$$

Параметр β определяет степень важности precision или recall. При $\beta \in (0, 1)$ приоритет отдается метрике precision, а при $\beta > 1$ больший вес оказывает recall. Чаще всего используется параметр $\beta = 1$, так как в этом случае влияние метрик становится сбалансированным. Такая мера называется F1-мерой, и ее формула имеет следующий вид:

$$F_1 = 2\frac{precision \cdot recall}{precision + recall}.$$

Глава 2. Результаты

2.1. Датасет

Для решения задачи классификации необходимо собрать данные для обучения модели. Данные собирались на аналитической платформе Etherscan, которая позволяет исследовать и анализировать транзакции, адреса, токены, цены и различные действия, происходящие в Ethereum. Также Etherscan содержит активный список размеченных адресов Ethereum [32]. В эту базу входят как адреса, связанные с конкретными продуктами, так и более общие группы, характеризующие поведение адресов в сети.

Среди последних были выделены 4 самые крупные поведенческие категории адресов:

- Token Contract (2287 адресов);
- Exchange (217 адресов);
- ICO Wallets (172 адреса);
- Mining (62 адреса).

Именно эти группы будут выступать в качестве базовых классов, для обучения модели.

Транзакции в сети Ethereum сдержат следующую информацию:

- Время транзакции;
- Адрес отправителя;
- Адрес получателя;
- Количество пересланного ETH;
- Количество затраченного на отправку ETH.

Также Etherscan позволяет определить, является ли адрес смарт-контрактом.

Из списка транзакций, содержащих данную информацию, необходимо выделить метрики, на основе которых будет производиться классификация. В качестве таких признаков были выделены:

1. Время между транзакциями
    a. Среднее (AVG_TIME_BETWEEN_TRANS)
    b. Среднеквадратичное отклонение (DEVIATION_TIME_BETWEEN_TRANS)
2. Время между исходящими транзакциями
    a. Среднее (AVG_TIME_BETWEEN_SENT_TRANS)
    b. Среднеквадратичное отклонение (DEVIATION_TIME_BETWEEN_SENT_TRANS)
3. Время между входящими транзакциями
    a. Среднее (AVG_TIME_BETWEEN_RECEIVED_TRANS)
    b. Среднеквадратичное отклонение (DEVIATION_TIME_BETWEEN_RECEIVED_TRANS)
4. Количество ETH, участвующего в транзакции
    a. Среднее (AVG_TRANS_ETH)
    b. Среднеквадратичное отклонение (DEVIATION_TRANS_ETH)
5. Количество отправленного ETH
    a. Среднее (AVG_ETH_SENT)
    b. Среднеквадратичное отклонение (DEVIATION_ETH_SENT)
6. Количество полученного ETH
    a. Среднее (AVG_ETH_RECEIVED)

b. Среднеквадратичное отклонение (DEVIATION_ETH_RECEIVED)

7. Процент транзакций с участием смарт-контрактов (PERCENT_OF_SMART_CONTRACT_TRANS)
8. Процент транзакций, полученных от смарт-контрактов (PERCENT_OF_TRANS_RECEIVED_FROM_SMART_CONTRACTS)
9. Процент транзакций, отправленных смарт-контрактам (PERCENT_OF_TRANS_SENT_TO_SMART_CONTRACTS)
10. Процент ETH, участвовавшего в транзакциях со смарт-контрактами (PERCENT_OF_SMART_CONTRACT_ETH)
11. Процент ETH, полученного от смарт-контрактов (PERCENT_OF_ETH_RECEIVED_FROM_SMART_CONTRACTS)
12. Процент ETH, отправленного смарт-контрактам (PERCENT_OF_ETH_SENT_TO_SMART_CONTRACTS)

Ресурс Etherscan не предоставляет открытого API для работы, поэтому было принято решение использовать инструмент для автоматизации действий веб-браузера Selenium [33] для парсинга данных. Программа для парсинга была написана на языке Java. Выбор языка обусловлен тем, что Selenium написан на Java и работает наилучшим образом именно в Java приложениях, ввиду нативности.

Программа для формирования датасета состоит из 2 основных элементов:

- DataLoader – формирует списки транзакций для каждого адреса из 4 групп в CSV формате.
- FeatureExtractor – выделяет описанные выше признаки из списков транзакций и формирует датасет в JSON формате.

Некоторые адреса имеют небольшое количество транзакций, что ставит под сомнение валидность извлеченных метрик. Для этого был введен параметр $threshold = 100$, отсеивающий адреса, содержащие меньше 100 транзакций.

В результате был сформирован датасет, содержащий 538 адресов.

Чтобы убедиться в том, что выделенные метрики являются независимыми, была построена матрица корреляции [34]:

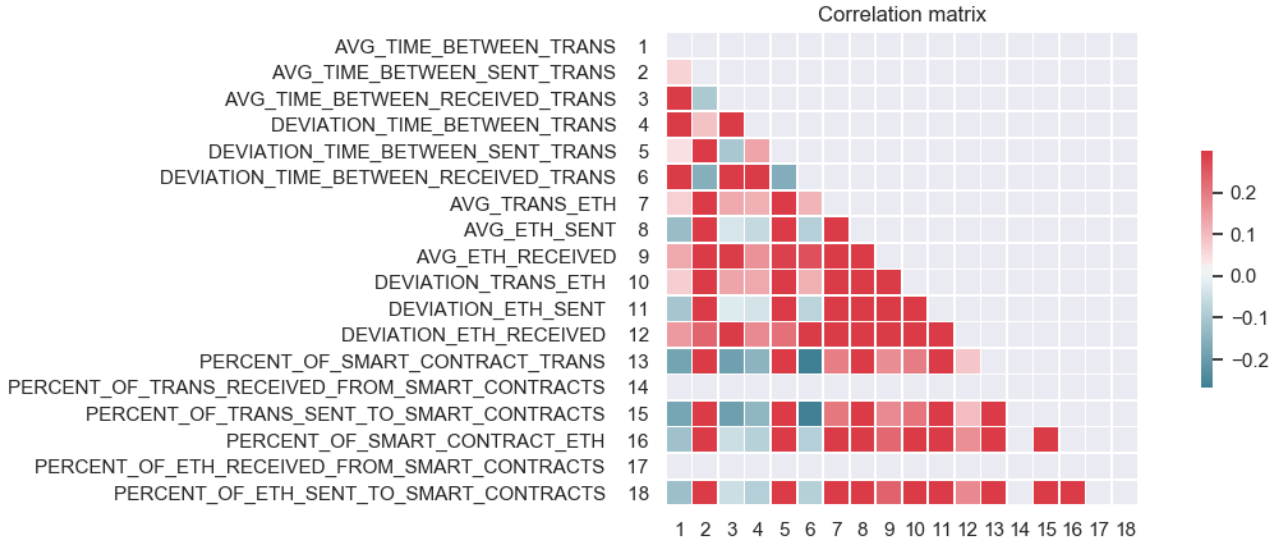


Рис. 13. Матрица корреляции признаков.

Корреляционный анализ (рис. 13) показывает, что зависимости между признаками незначительные (порядка 0.2), значит имеет смысл учитывать все метрики в обучении.

2.2. Результаты классификации

Вопрос сравнения работы классификаторов даже в рамках одной задачи имеет ряд проблем, связанных с разбиением датасета на обучающие и тестовые множества. Одна и та же модель будет давать различную точность, в зависимости от разбиения данных. Строгая фиксация обучающей и тестовой выборок в общем случае также некорректна, ввиду различия алгоритмов классификации. Каждая модель учитывает различные метрики с тем или иным приоритетом. Поэтому сравнение проводилось на большом числе разбиений датасета, чтобы наиболее адекватно оценить работу классификаторов в рамках данной задачи. Для каждого метода приведены графики работы алгоритмов для 6 различных разбиений. Данные результаты наиболее показательны и в полной мере охватывают весь диапазон проведенных экспериментов.

Для решения поставленной задачи был выбран язык Python 3, так как для него разработано большое количество библиотек для классификации и визуализации данных. Модели классификаторов создавались с использованием библиотеки машинного обучения Scikit-learn [35], так как она предоставляет большой выбор алгоритмов обучения с учителем и имеет хорошую документацию. Нейронная сеть проектировалась с использованием библиотеки Keras [36]. Библиотека предоставляет удобный API поверх других различных нейросетевых фреймворков (таких как TensorFlow, Theano, Microsoft Cognitive Toolkit и других). В качестве бэкенда использовалась библиотека TensorFlow [37] от компании Google. Она является одной из самых популярных нейросетевых библиотек, благодаря своей гибкости и производительности. Также библиотека включает инструмент мониторинга TensorBoard [38], визуализирующий процесс обучения сети.

2.2.1. Метод k-ближайших соседей

Работа данного классификатора была проверена на нескольких различных разбиениях датасета. Максимальная точность достигнута с параметром $k = 5$ соседей. Различные метрики результатов классификации представлены ниже (рис. 14 – 17).

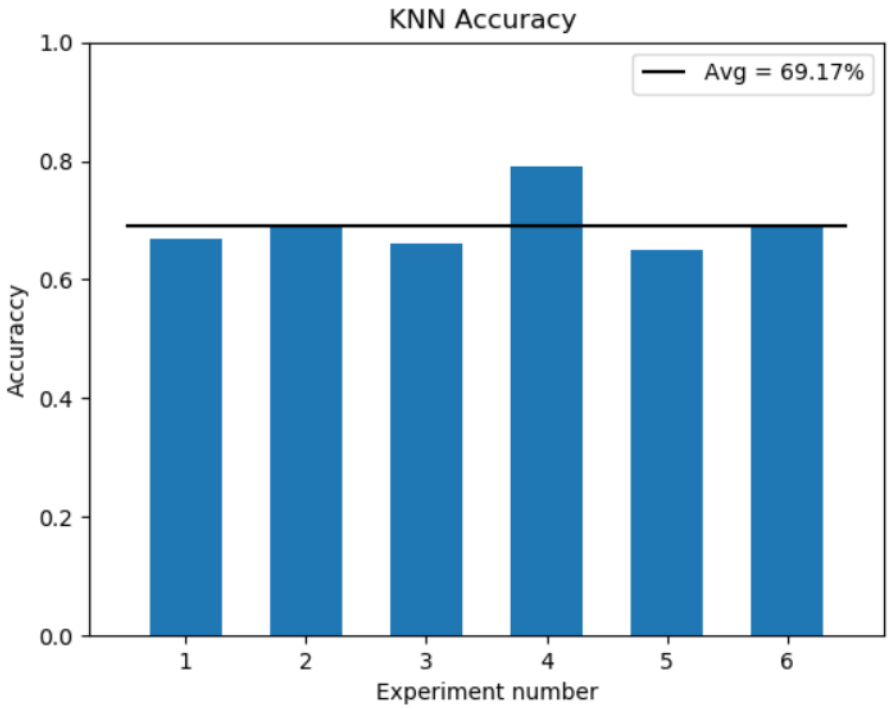


Рис. 14. Диаграмма точности работы метода k-ближайших соседей.

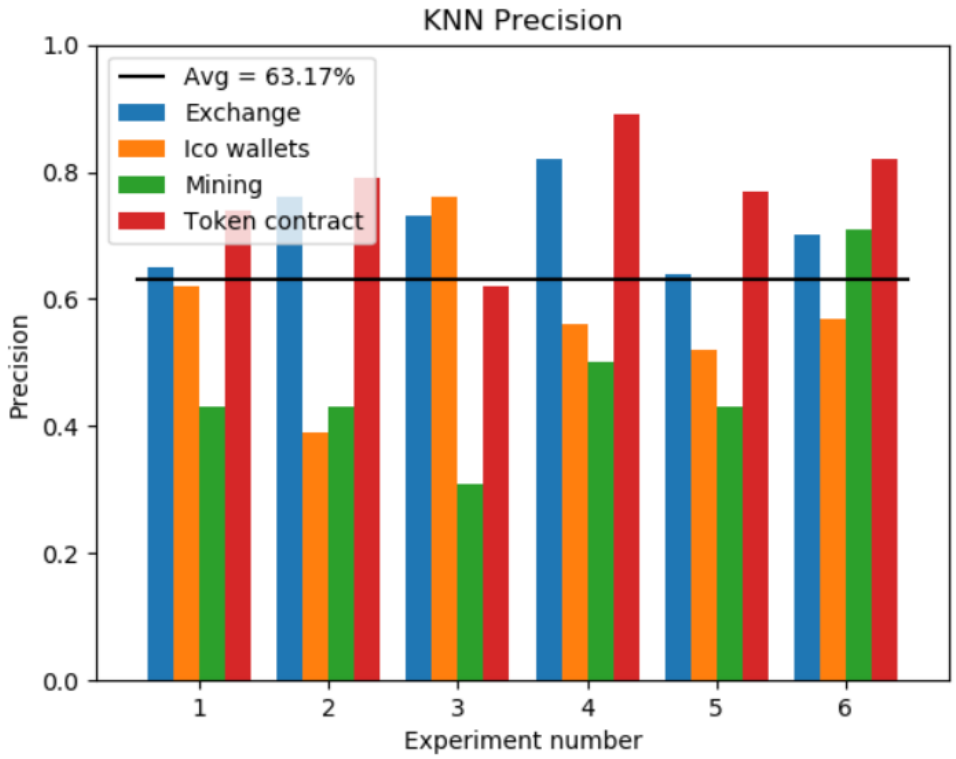


Рис. 15. Диаграмма значений precision метода k-ближайших соседей.

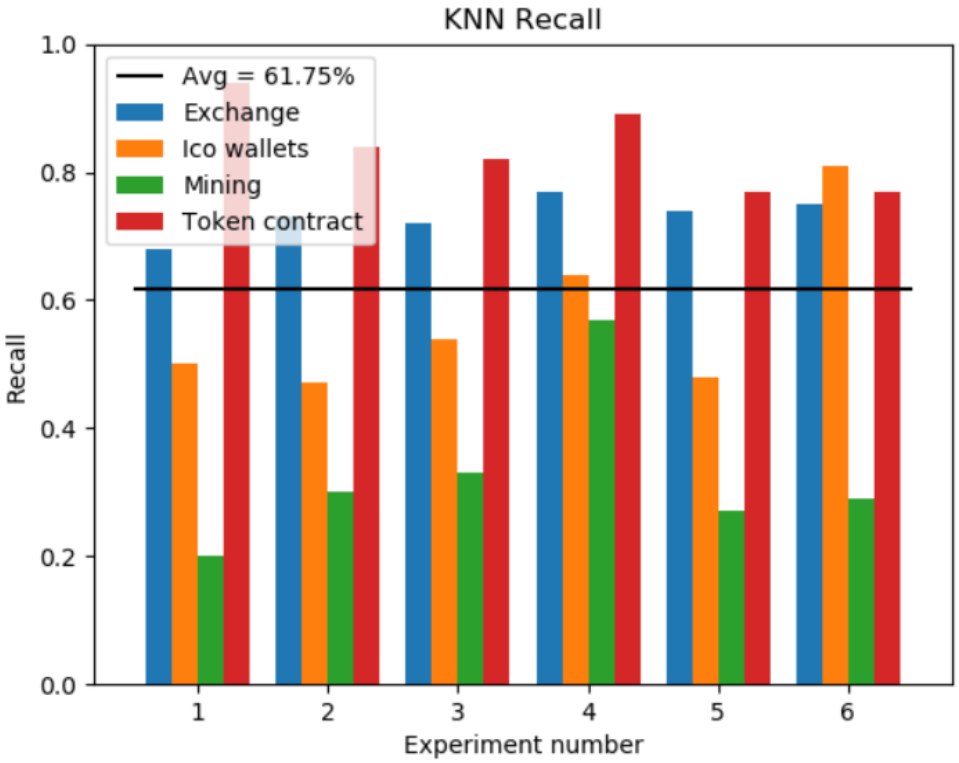


Рис. 16. Диаграмма значений recall метода k-ближайших соседей.

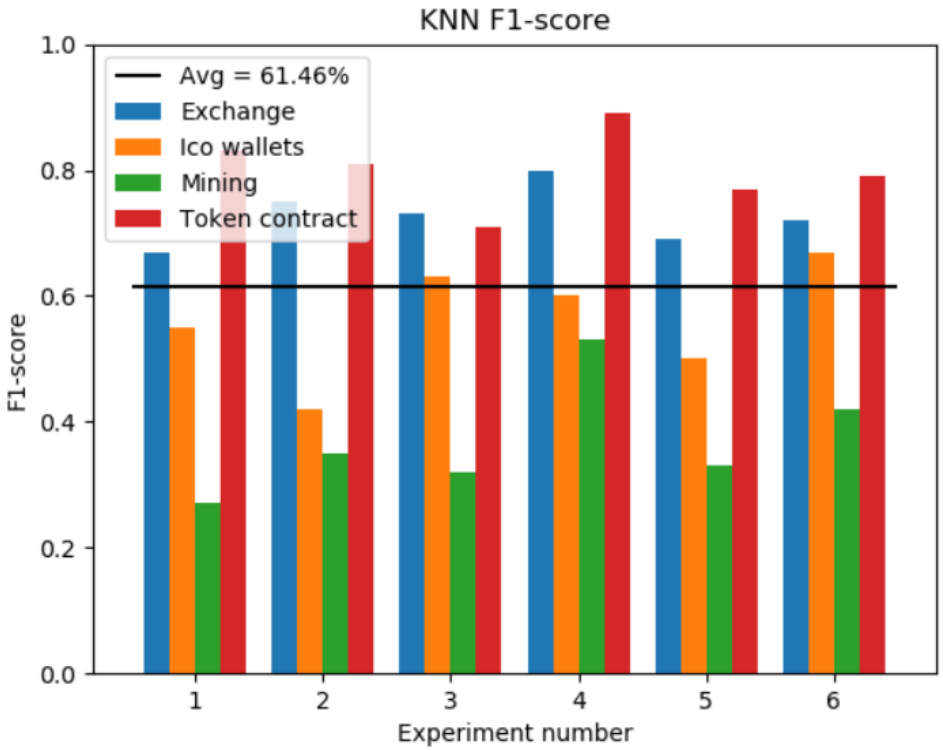


Рис. 17. Диаграмма значений F1-меры для метода k-ближайших соседей.

Средняя точность работы данного классификатора составила всего 69.17%, чего явно недостаточно. Из диаграмм (рис. 15 – 17) видно, что большинство ошибок классификации было совершено на классе “Mining”.

Для каждого разбиения датасета был проведен анализ важности признаков. Из выборки последовательно исключалось по одной метрике и проводилась валидация на тестовом множестве. На графиках (рис. 18) представлена зависимость точности классификатора от номера исключенной метрики. Черная горизонтальная линия показывает точность на исходном датасете.

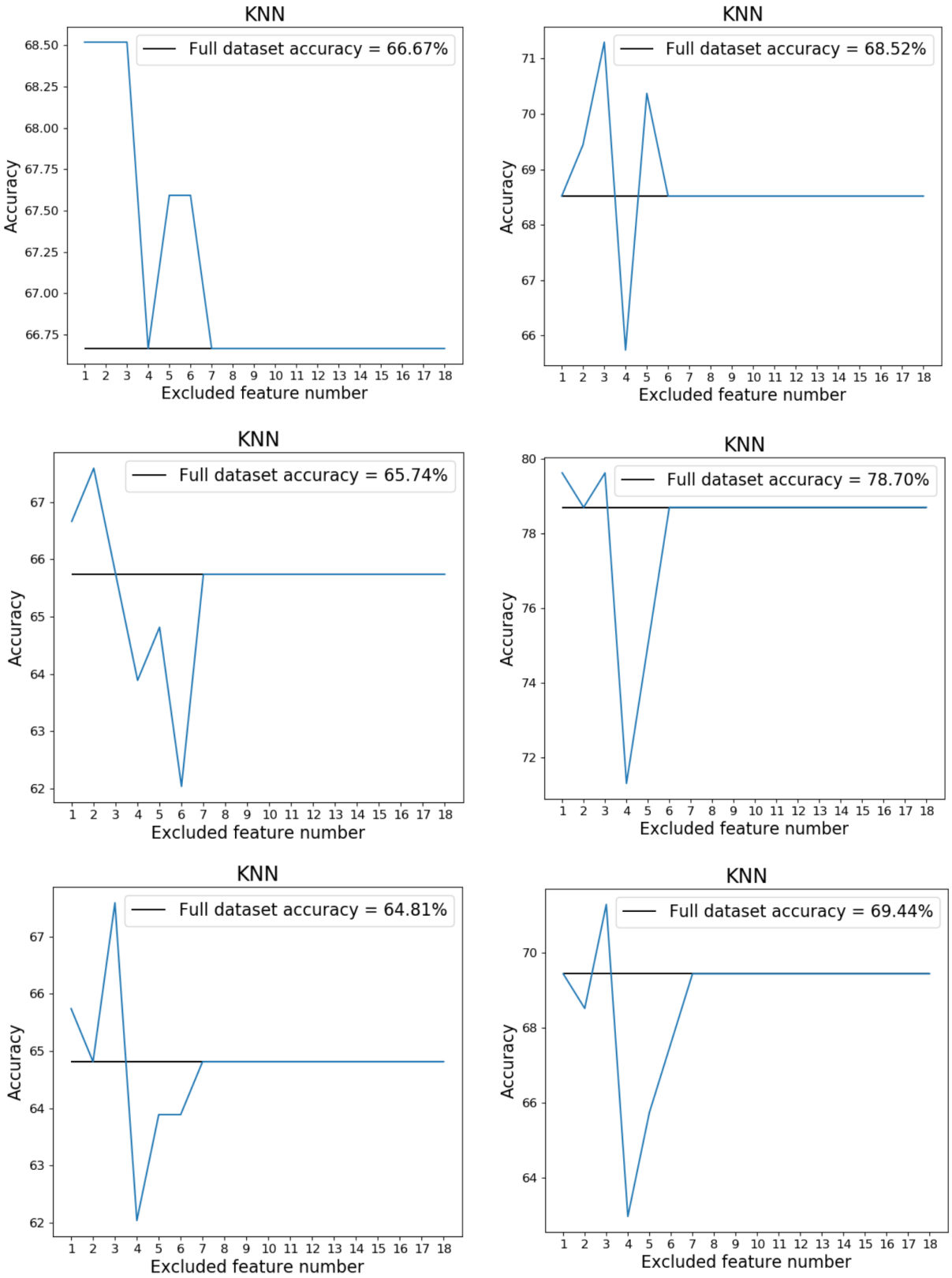
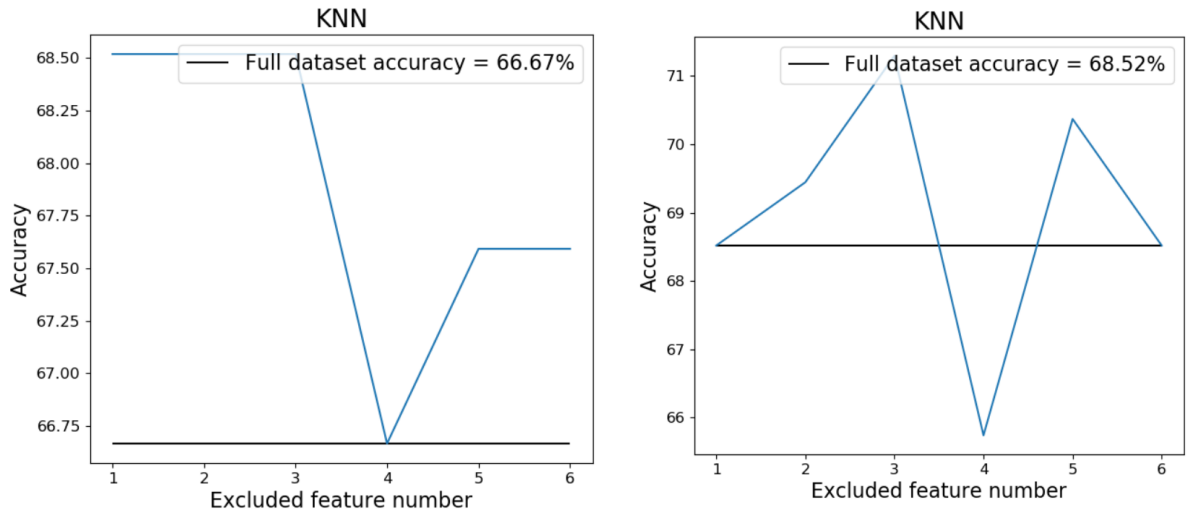


Рис. 18. Графики зависимости точности классификации методом k-ближайших соседей от номера исключенной из датасета метрики.

Из графиков видно, что на точность классификации влияют только первые 6 признаков:

1. AVG_TIME_BETWEEN_TRANS,
2. AVG_TIME_BETWEEN_SENT_TRANS,
3. AVG_TIME_BETWEEN_RECEIVED_TRANS,
4. DEVIATION_TIME_BETWEEN_TRANS,
5. DEVIATION_TIME_BETWEEN_SENT_TRANS,
6. DEVIATION_TIME_BETWEEN_RECEIVED_TRANS.

Результаты работы классификатора на обновленном датасете приведены ниже (рис. 19).



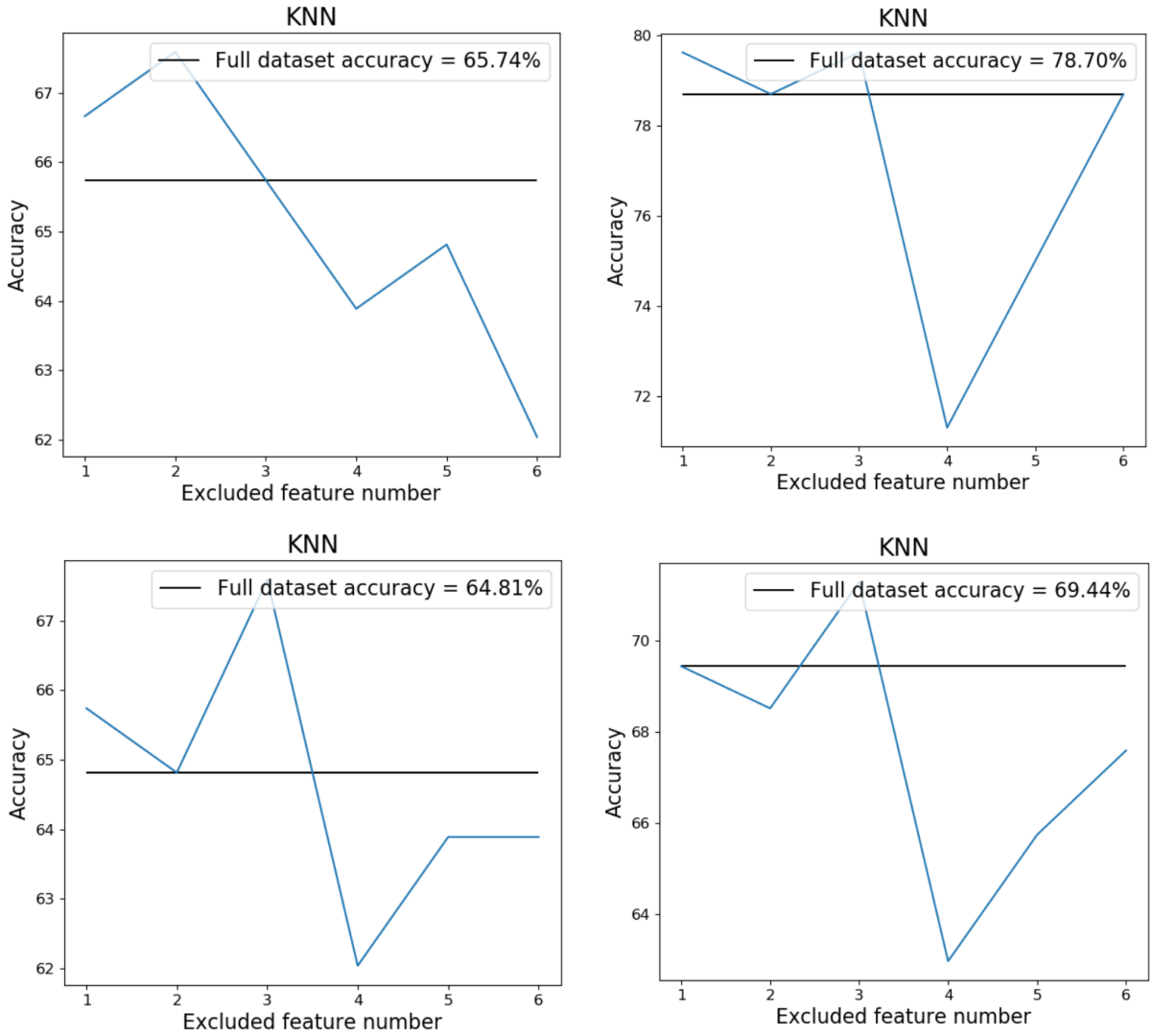
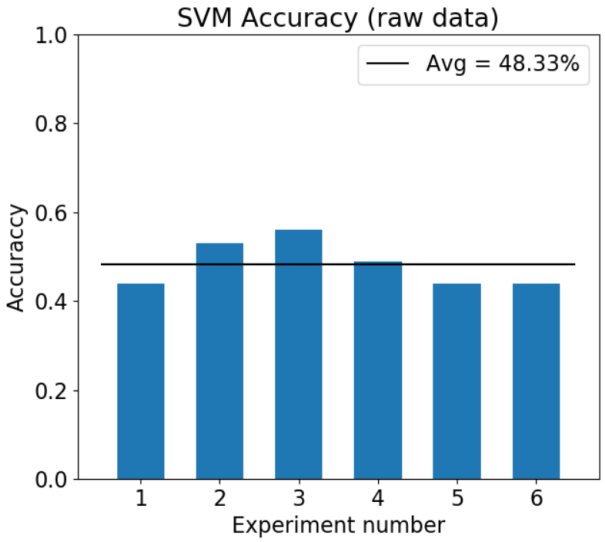


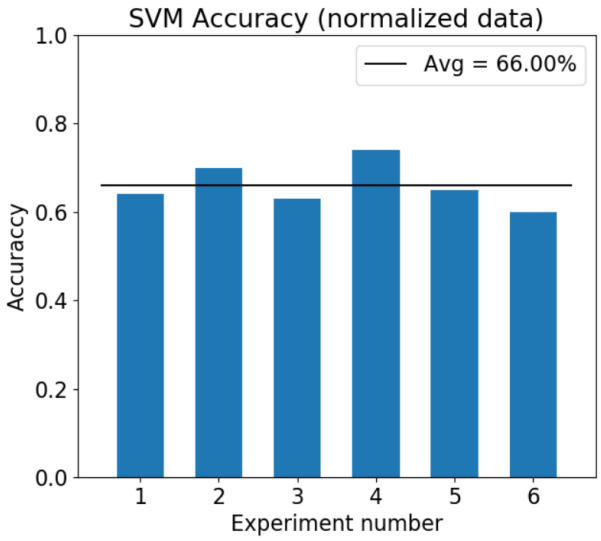
Рис. 19. Графики зависимости точности классификации методом k-ближайших соседей от номера исключенной из обновленного датасета метрики.

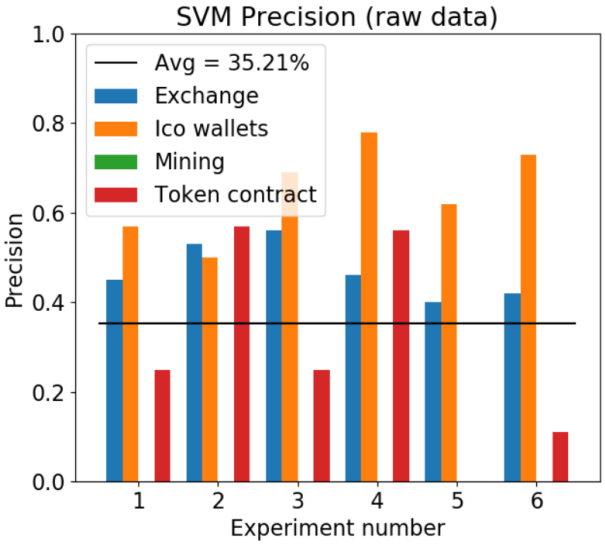
По графикам (рис. 19) видно, что точность классификации не изменилась.

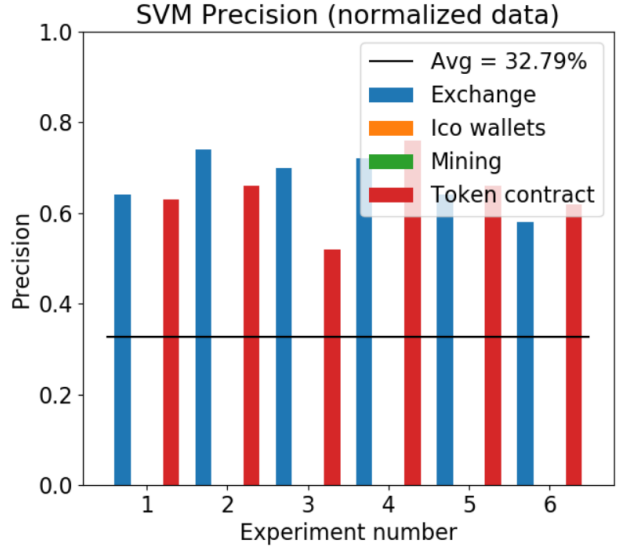
2.2.2. Метод опорных векторов

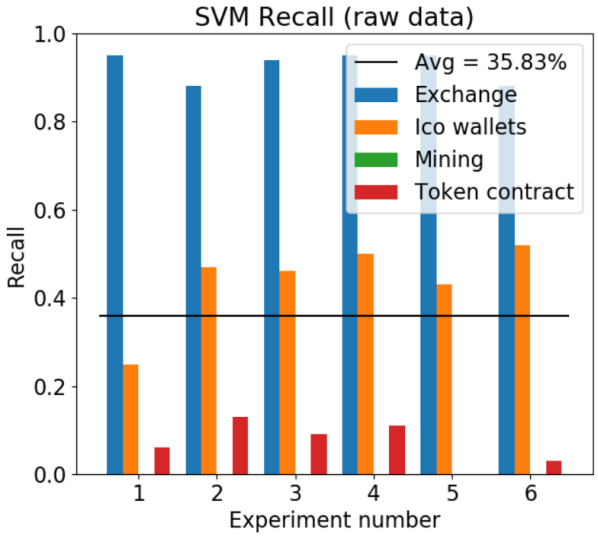
Работа метода опорных векторов проверялась на исходных и на нормализованных данных, метрики оценки работы представлены на рисунке 20. В качестве ядра использовалась радиально-базисная функция (radial basis function, RBF) [39].

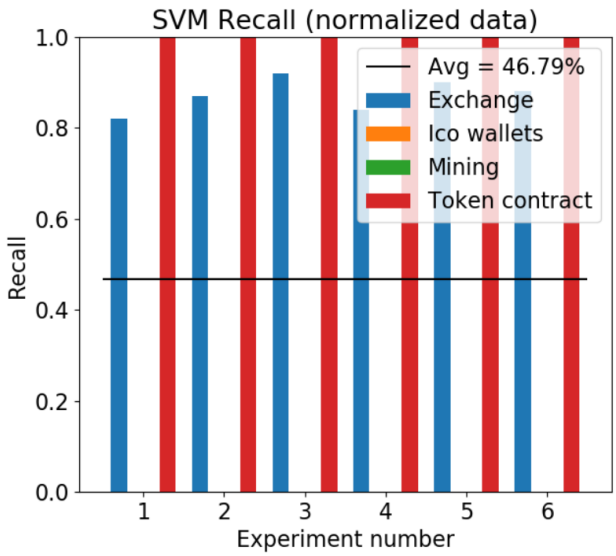

SVM Accuracy (raw data)
Avg = 48.33%
Accuracy
Experiment number


SVM Accuracy (normalized data)
Avg = 66.00%
Accuracy
Experiment number


SVM Precision (raw data)
Avg = 35.21%
Exchange
Ico wallets
Mining
Token contract
Precision
Experiment number


SVM Precision (normalized data)
Avg = 32.79%
Exchange
Ico wallets
Mining
Token contract
Precision
Experiment number


SVM Recall (raw data)
Avg = 35.83%
Exchange
Ico wallets
Mining
Token contract
Recall
Experiment number


SVM Recall (normalized data)
Avg = 46.79%
Exchange
Ico wallets
Mining
Token contract
Recall
Experiment number

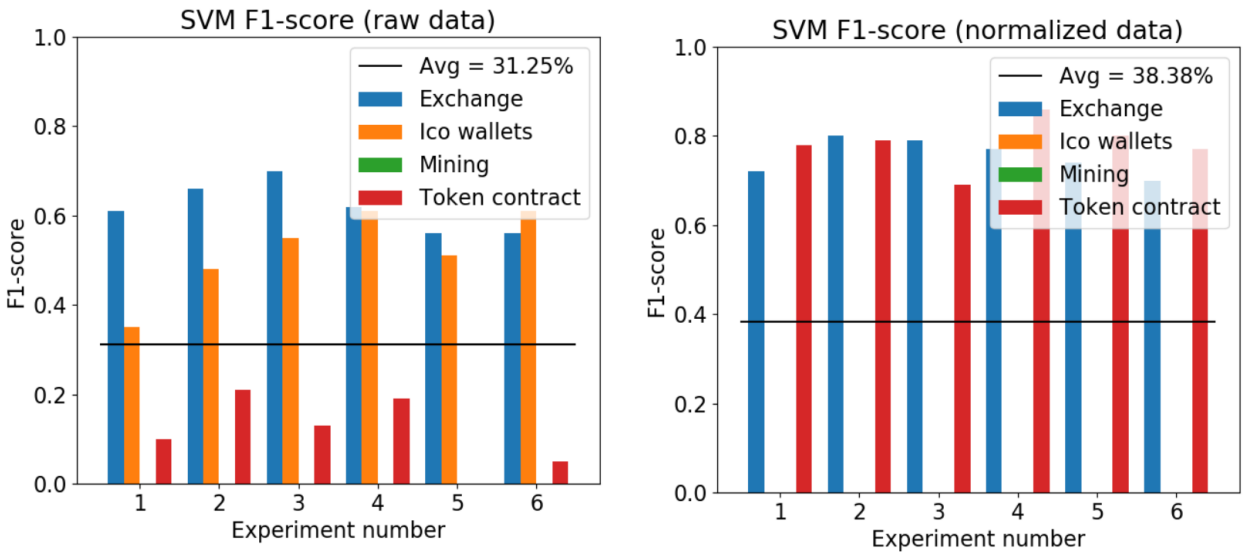


Рис. 20. Диаграммы точности, precision, recall и F1-меры для метода опорных векторов на исходных (слева) и нормализованных (справа) данных.

На исходных данных модель показала очень низкую точность и оказалась совершенно неспособной классифицировать объекты класса “Mining”. Нормализация данных позволила немного повысить общую точность распознавания, однако, модель утратила способность классификации “ICO wallets”. Варьирование параметров модели не дало существенных улучшений. Таким образом, метод оказался непригодным для решения поставленной задачи.

2.2.3. Решающие деревья

Решающие деревья показывают довольно неплохую точность классификации относительно предыдущих методов. В качестве функции качества ветвления в модели использовался критерий Джини. Метрики оценки классификации представлены на рисунках 21 – 24.

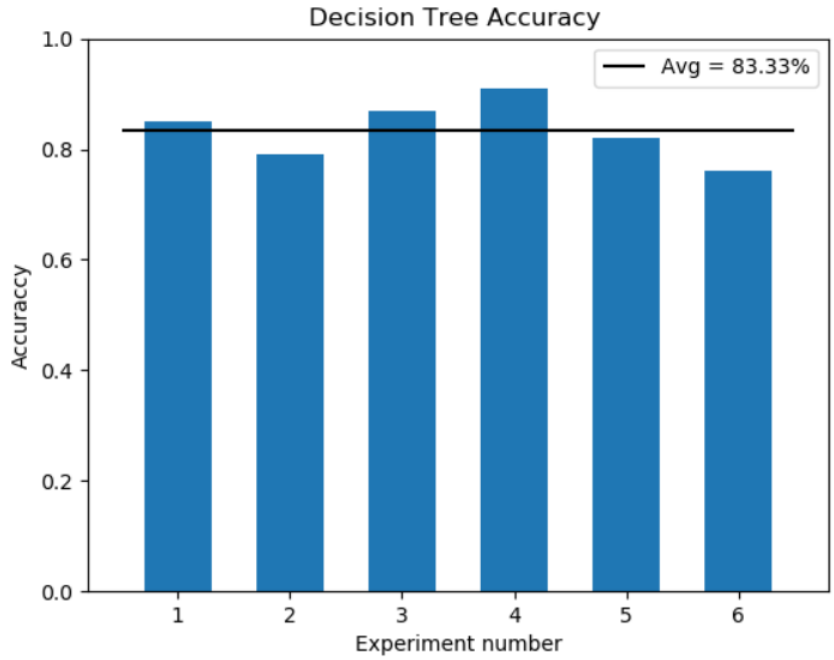


Рис. 21. Диаграмма точности работы дерева решений.

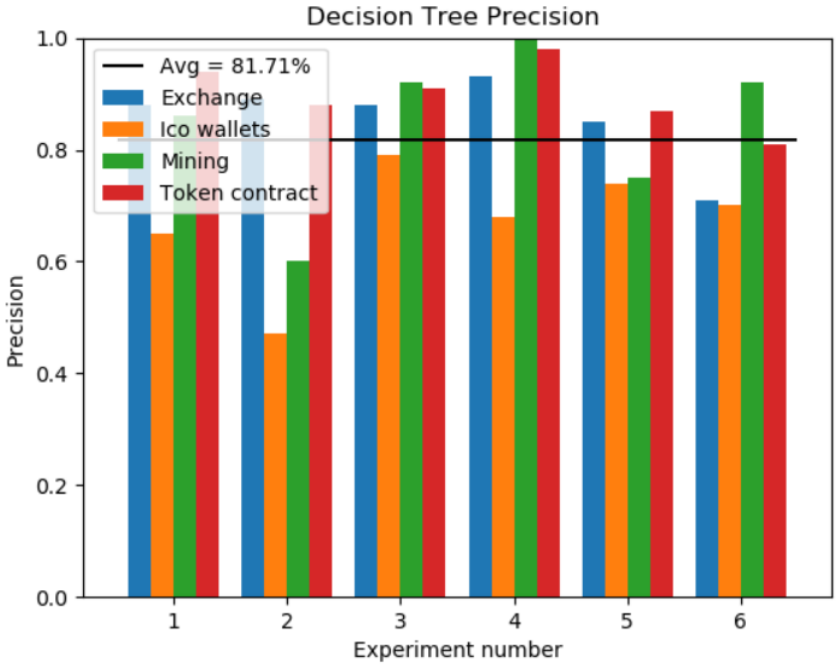


Рис. 22. Диаграмма значений precision дерева решений.

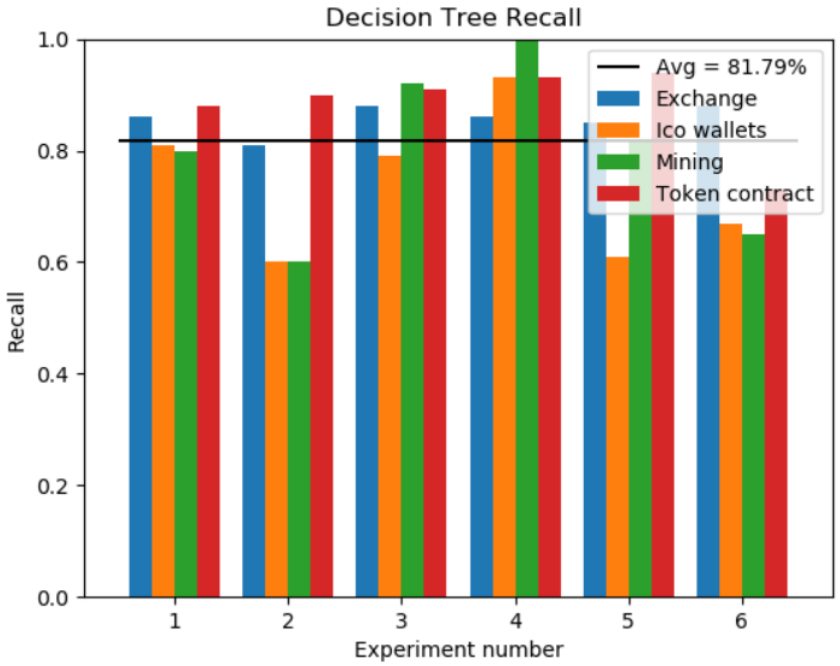


Рис. 23. Диаграмма значений recall дерева решений.

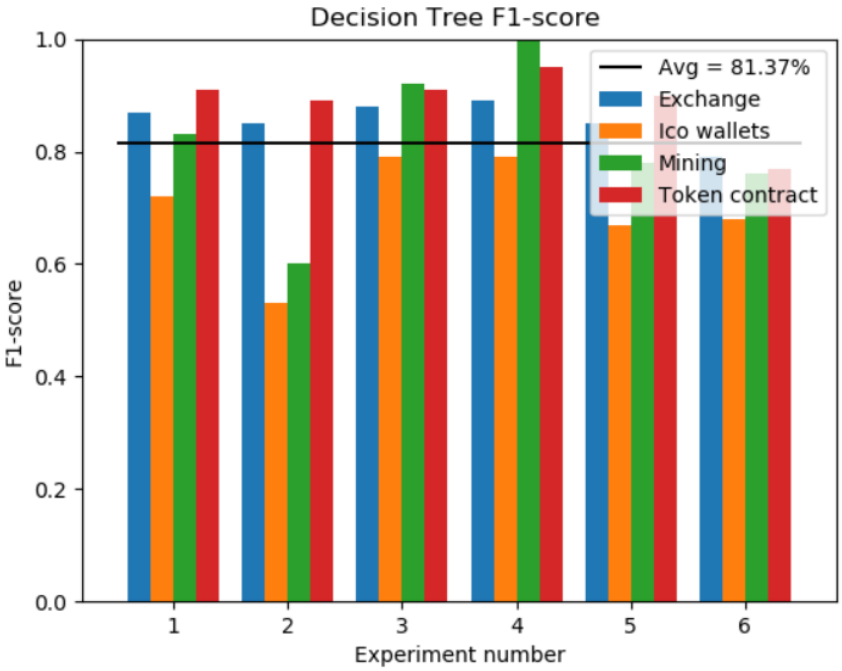


Рис. 24. Диаграмма значений F1-меры дерева решений.

На диаграммах ниже (рис. 25) представлены важности признаков (feature importance) для различных разбиений датасета.

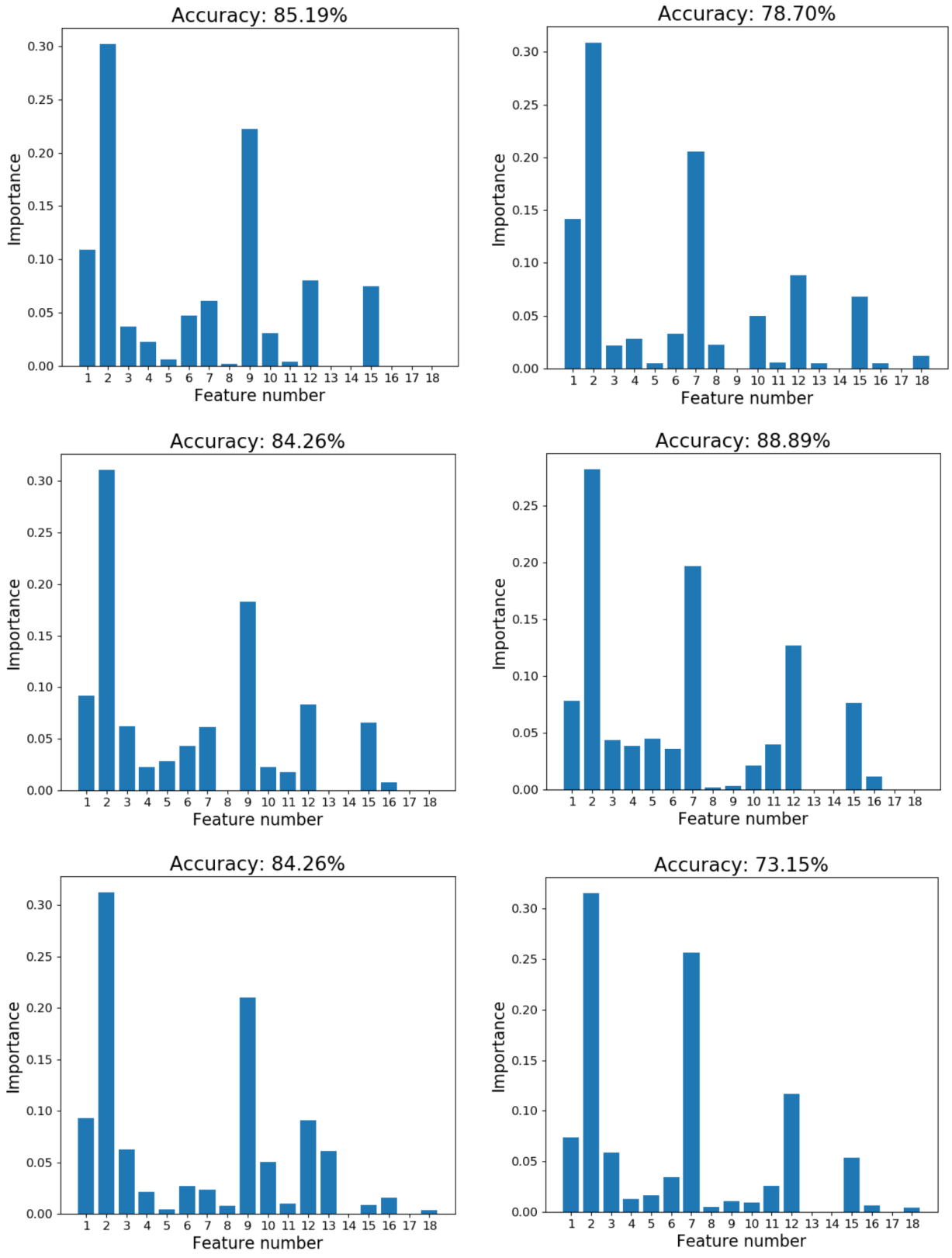


Рис. 25. Feature importance дерева решений.

Видно, что в основной вклад в классификацию вносят следующие признаки:

1. AVG_TIME_BETWEEN_TRANS (1),
2. AVG_TIME_BETWEEN_SENT_TRANS (2),
3. AVG_TRANS_ETH (7),
4. AVG_ETH_RECEIVED (9),
5. DEVIATION_ETH_RECEIVED (12),
6. PERCENT_OF_TRANS_SENT_TO_SMART_CONTRACTS (15).

2.2.4. Случайный лес

Модель случайного леса формировалась из 100 решающих деревьев. Точность классификации данного метода (рис. 26) в среднем выше, чем у одного дерева, что вполне ожидаемо. Также precision (рис. 27), recall (рис. 28) и F1-мера (рис. 29) показывают вполне приемлемые значения точности для каждого класса. Это говорит о том, что классификатор можно применять для решения задачи.

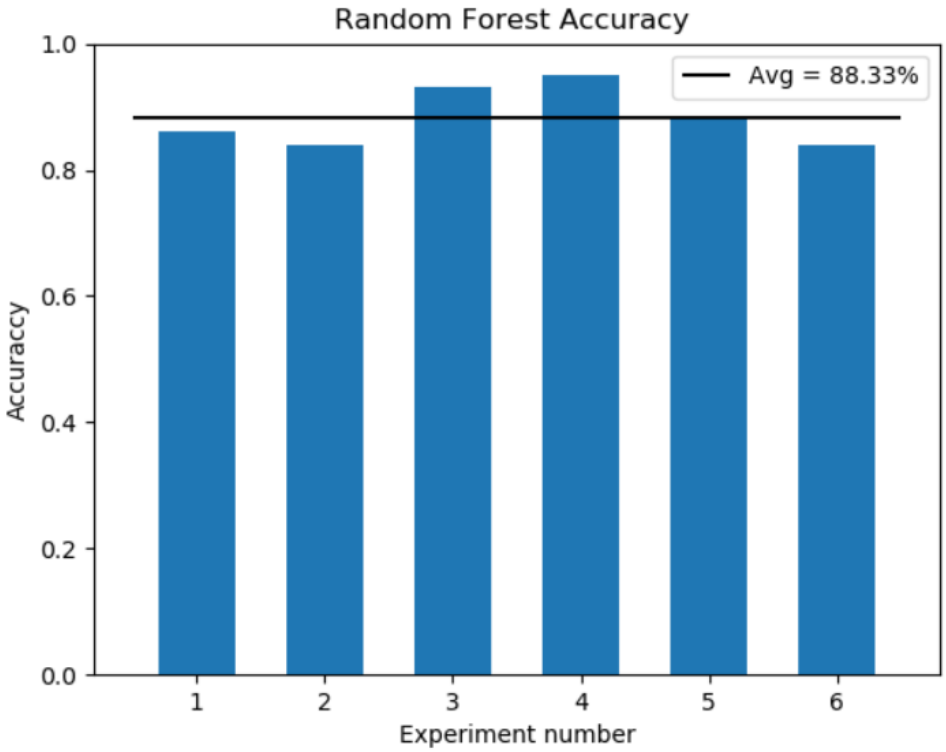


Рис. 26. Диаграмма точности работы случайного леса.

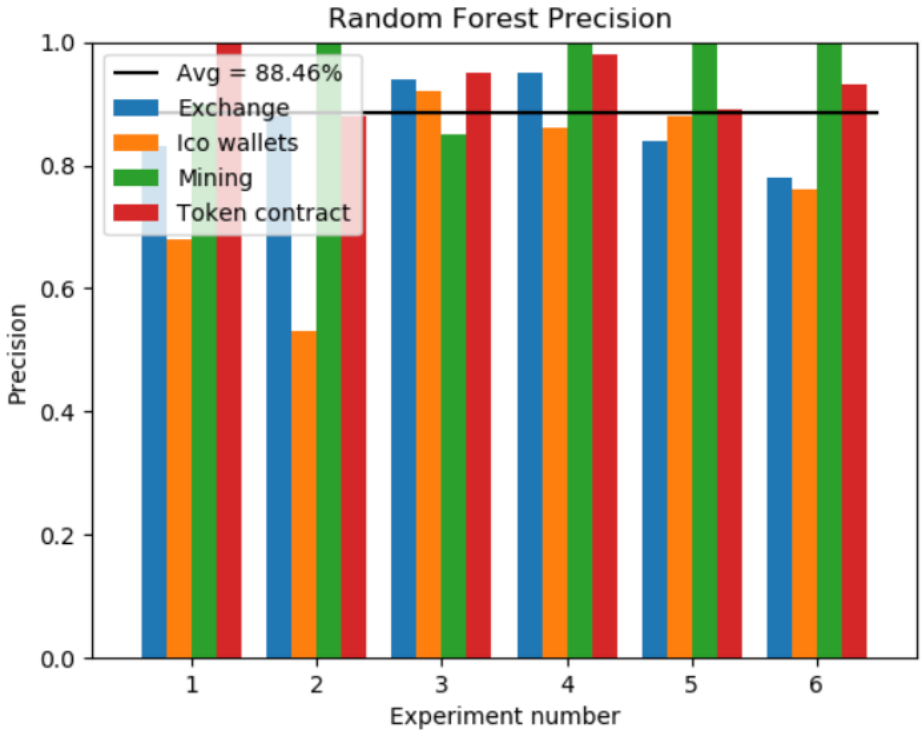


Рис. 27. Диаграмма значений precision случайного леса.

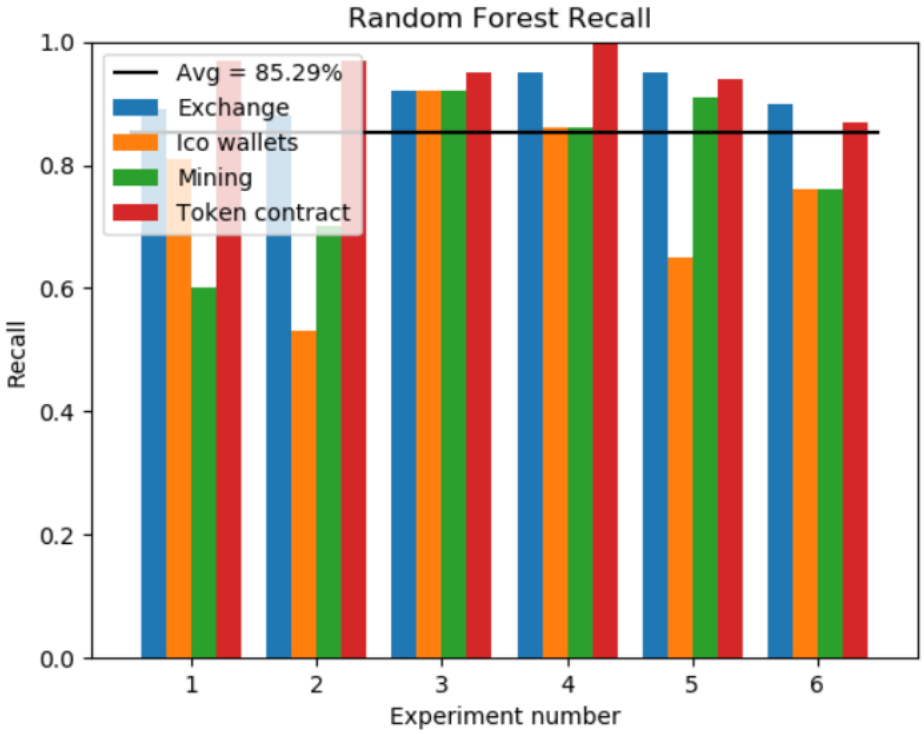


Рис. 28. Диаграмма значений recall случайного леса.

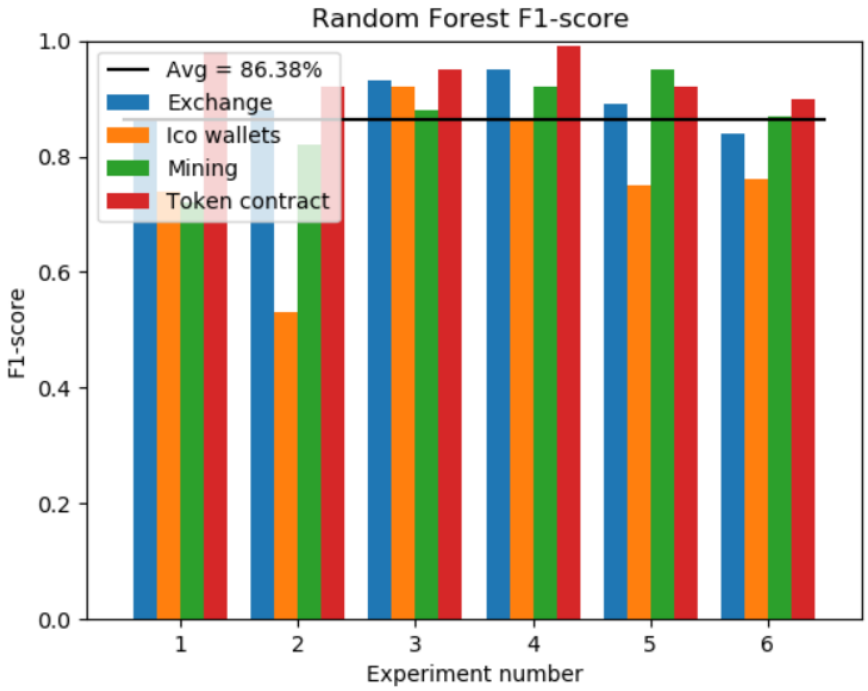


Рис. 29. Диаграмма значений F1-меры случайного леса.

Feature importance данного метода приведены ниже (рис. 30).

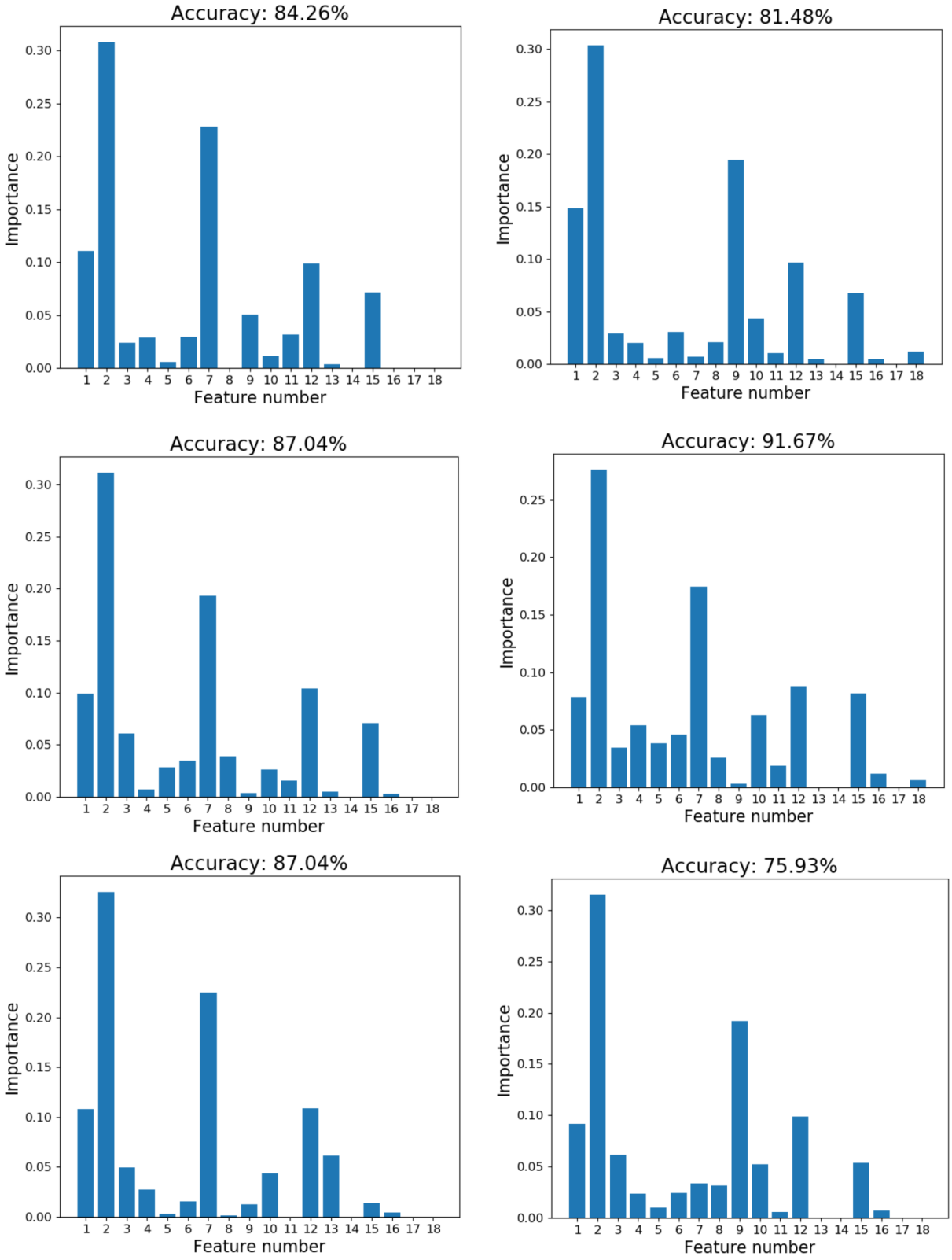


Рис. 30. Feature importance случайного леса.

Важности признаков случайного леса по своему характеру схожи с решающими деревьями. Наиболее значимые признаки остаются теми же самыми.

2.2.5. Градиентный бустинг

Модель градиентного бустинга обучалась в течение 100 итераций. В качестве функции потерь использовалось функция полиномиального отклонения (multinomial deviance loss function) [40].

Как и случайный лес, градиентный бустинг показывает хорошие результаты работы по сравнению с другими методами классификации, однако, немного уступает случайному лесу по всем метрикам (рис. 31 – 34).

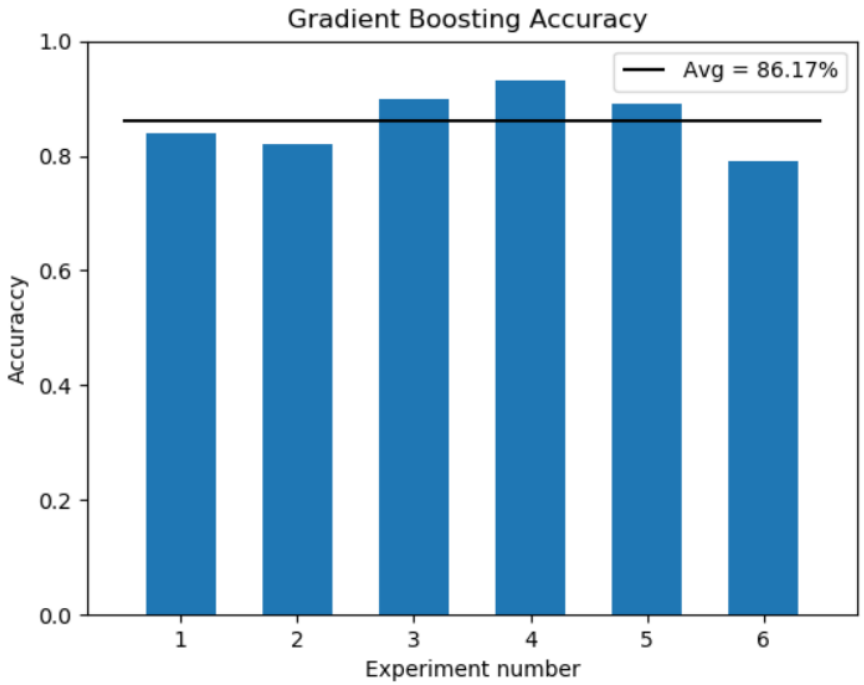


Рис. 31. Диаграмма точности работы градиентного бустинга.

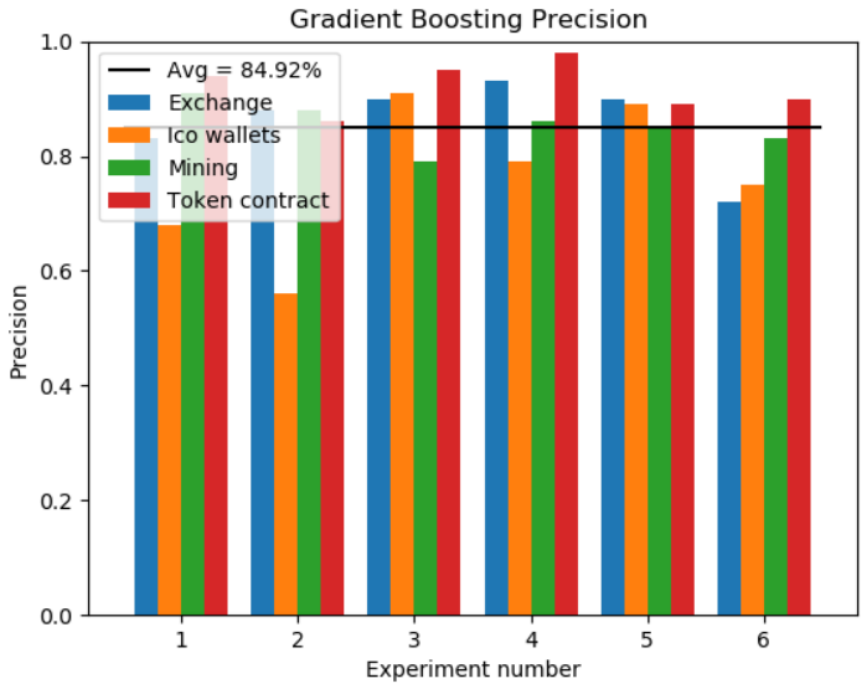


Рис. 32. Диаграмма значений precision градиентного бустинга.

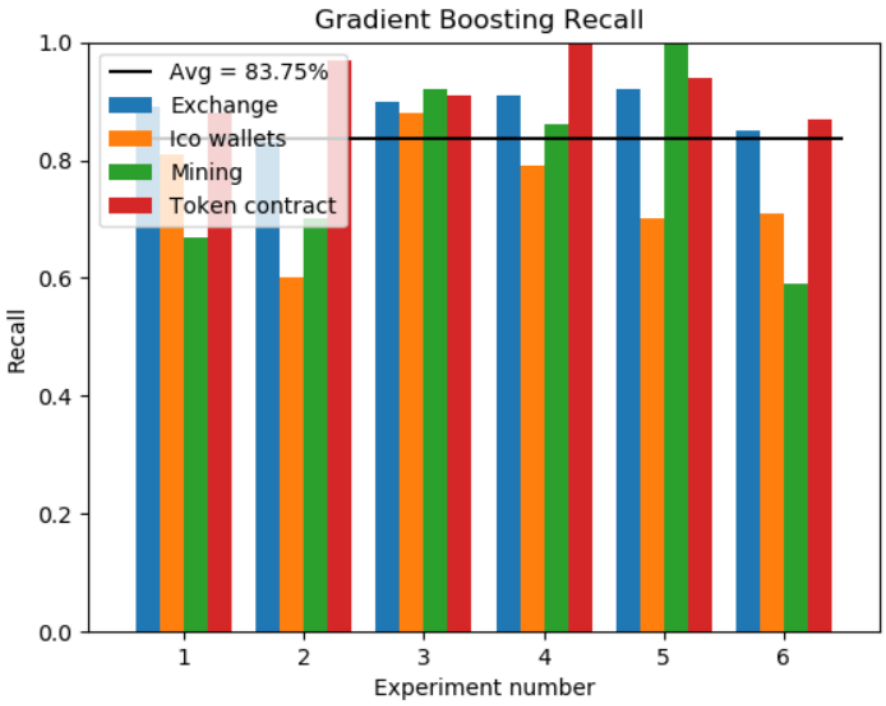


Рис. 33. Диаграмма значений recall градиентного бустинга.

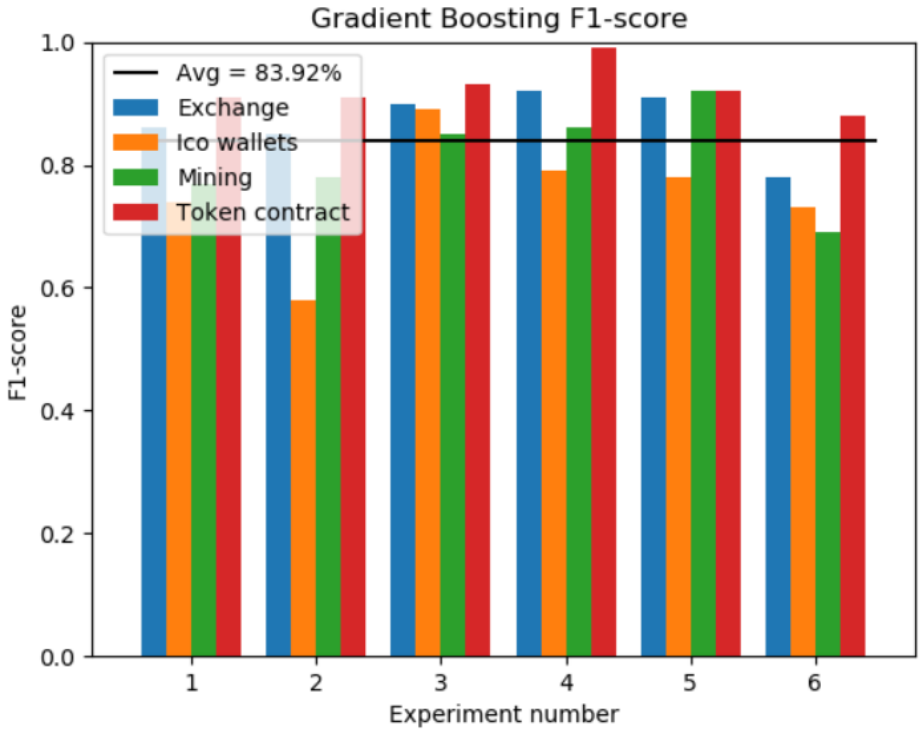
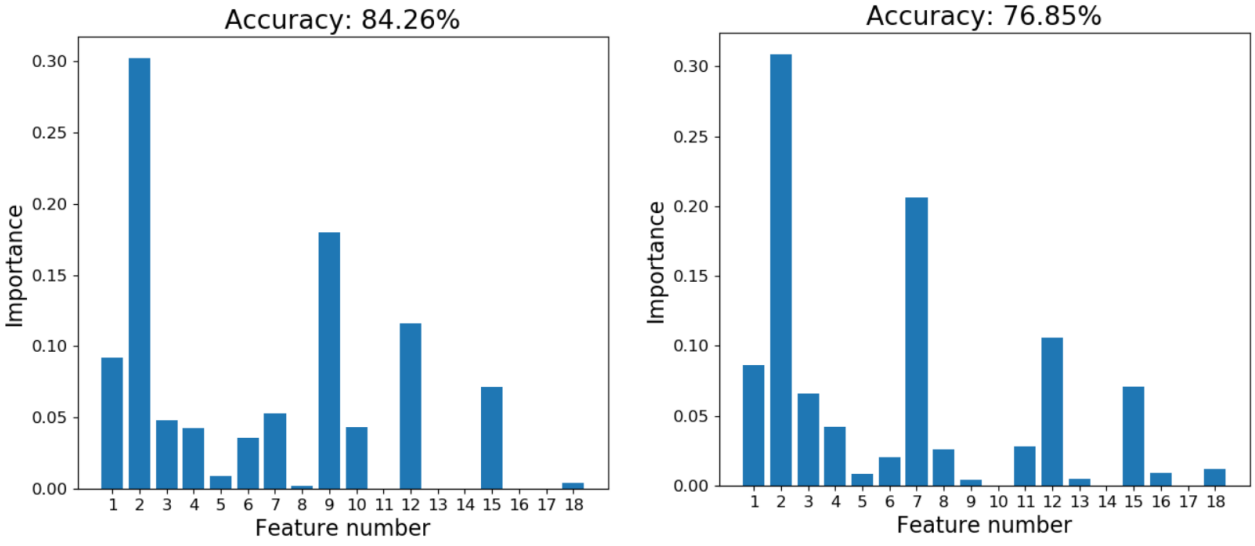


Рис. 34. Диаграмма значений F1-меры для градиентного бустинга.

Диаграммы feature importance для градиентного бустинга представлены на рисунке 35.



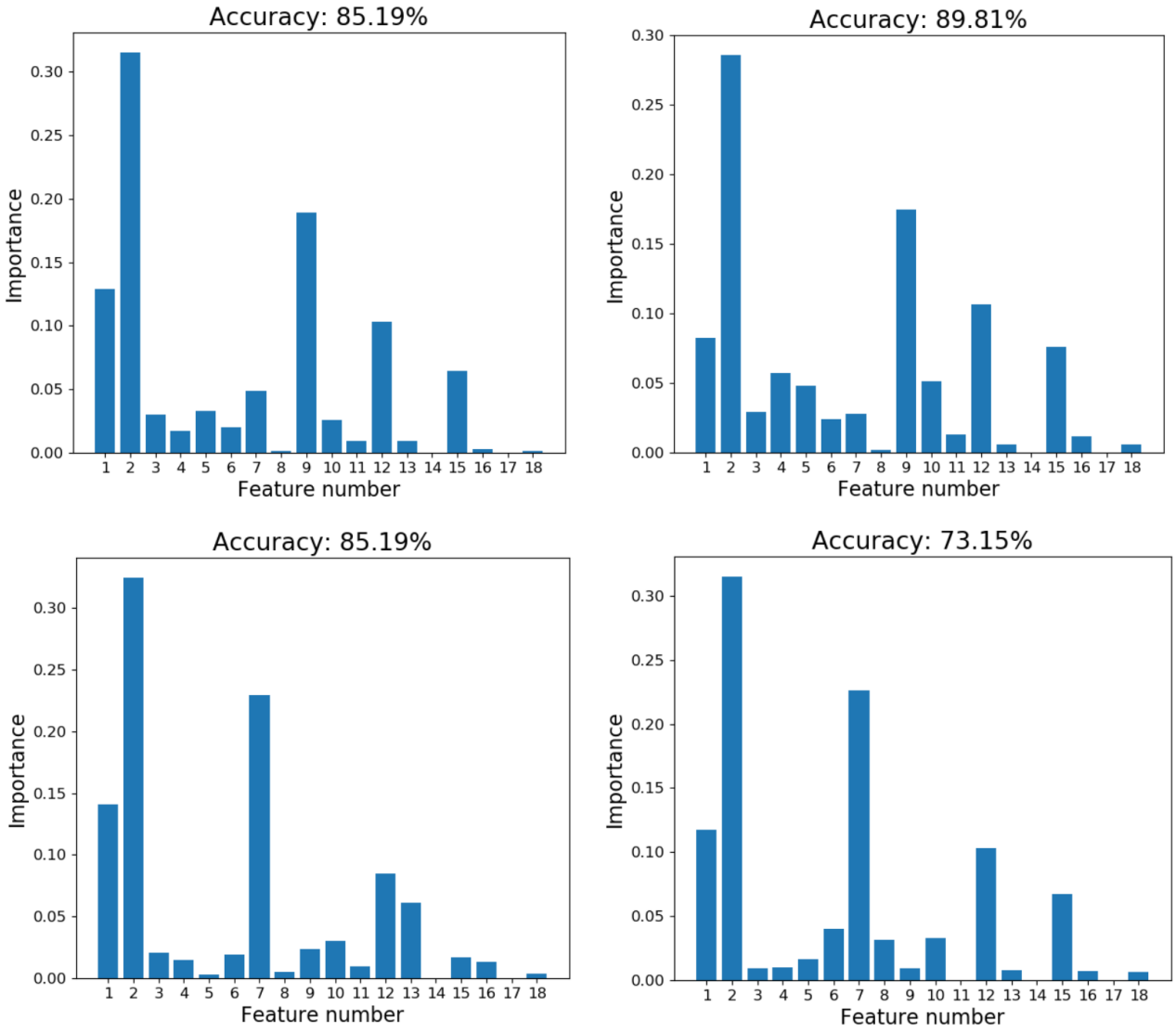


Рис. 35. Feature importance градиентного бустинга.

Значимость признаков метода (рис. 35) имеет аналогичный характер с двумя предыдущими методами (рис. 25, 30), что не удивительно, так как деревья решений лежат в основе работы случайного леса и градиентного бустинга.

2.2.6. Искусственные нейронные сети

Все признаки датасета можно разделить на 3 группы, несущие информацию о различных характеристиках адресов:

Таблица 3.

Группы признаков датасета.

1 группа (transactions)	AVG_TIME_BETWEEN_TRANS
	AVG_TIME_BETWEEN_SENT_TRANS
	AVG_TIME_BETWEEN_RECEIVED_TRANS
	DEVIATION_TIME_BETWEEN_TRANS
	DEVIATION_TIME_BETWEEN_SENT_TRANS
	DEVIATION_TIME_BETWEEN_RECEIVED_TRANS
2 группа (ETH)	AVG_TRANS_ETH
	AVG_ETH_SENT
	AVG_ETH_RECEIVED
	DEVIATION_TRANS_ETH
	DEVIATION_ETH_SENT
	DEVIATION_ETH_RECEIVED
3 группа (smart contracts)	PERCENT_OF_SMART_CONTRACT_TRANS
	PERCENT_OF_TRANS_RECEIVED_FROM_SMART_CONTRACTS
	PERCENT_OF_TRANS_SENT_TO_SMART_CONTRACTS
	PERCENT_OF_SMART_CONTRACT_ETH
	PERCENT_OF_ETH_RECEIVED_FROM_SMART_CONTRACTS
	PERCENT_OF_ETH_SENT_TO_SMART_CONTRACTS

Архитектура нейронной сети (рис. 36) проектировалась таким образом, чтобы учесть индивидуальные особенности этих групп.

Сеть состоит из четырех основных элементов: трех модулей предобработки на входах и решающего персептрона на выходе. Каждая группа метрик подается на вход своему модулю предобработки для выделения локальных признаков.

Далее эти признаки агрегируются в слое *Concatenate*, путем последовательного склеивания выходов подсетей в один слой. Решающий модуль завершает классификацию и выдает на выходе вероятности принадлежности элемента к каждому из четырех классов.

Все четыре модуля представляют из себя персептроны с пакетной нормализацией (Batch Normalization) и дропаутом (Dropout).

Все нейроны сети используют сигмоиду в качестве функции активации, за исключением последнего слоя, в котором используется функция softmax. Данная функция позволяет получить на выходе вероятности принадлежности элемента к классу.

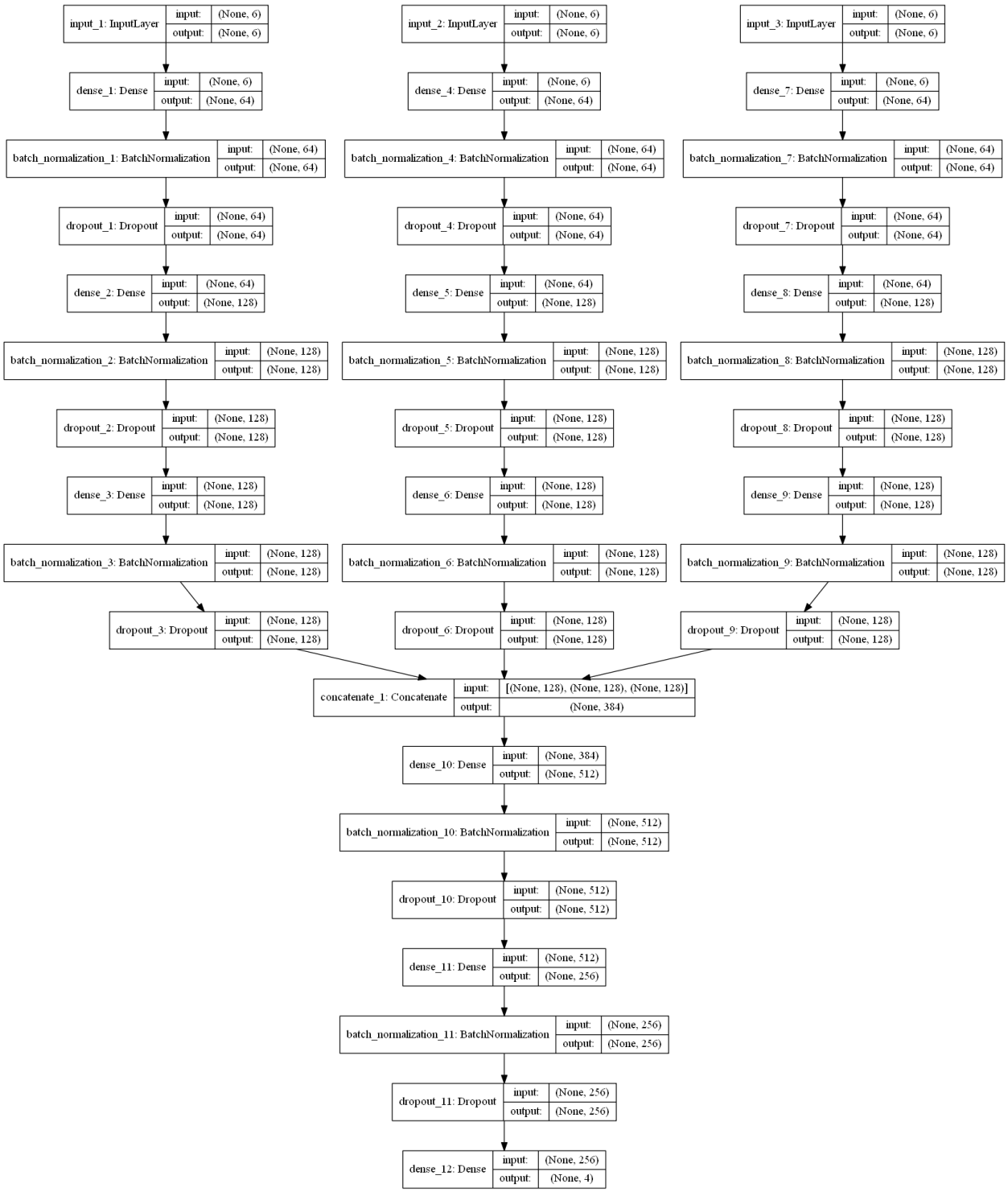


Рис. 36. Архитектура нейронной сети.

Обучение сети проводилось на 100 эпохах с параметром batch-size=32, в качестве функции потерь использовалась кросс-энтропия (categorical

crossentropy) [41]. Ниже представлены графики изменения точности (рис. 37) и функции потерь (рис. 38) в процессе обучения.

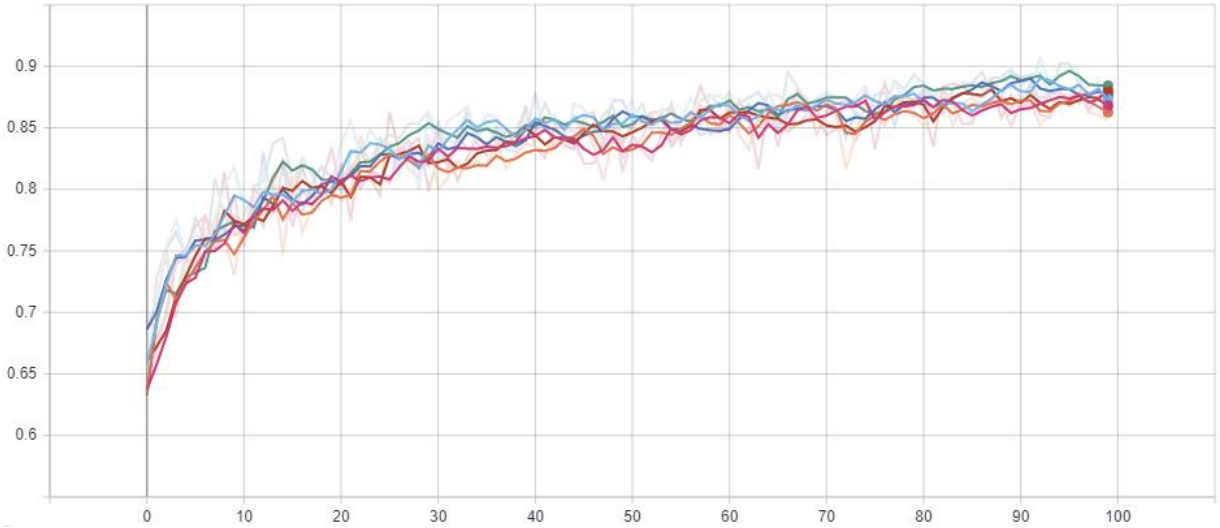


Рис. 37. График изменения точности нейронной сети в процессе обучения.

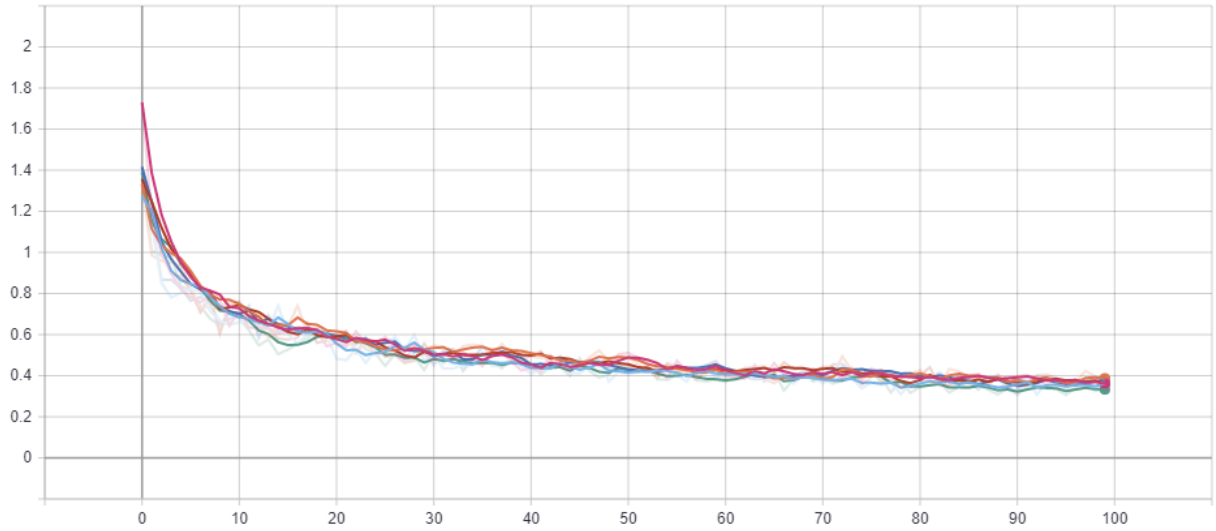


Рис. 38. График изменения значения функции потерь нейронной сети в процессе обучения.

Нейронная сеть показала хорошую общую точность на тестовом множестве (рис. 39), однако по диаграммам (рис. 40 – 42) видно, что точность определения класса “ICO Wallets” оказалась недостаточной.

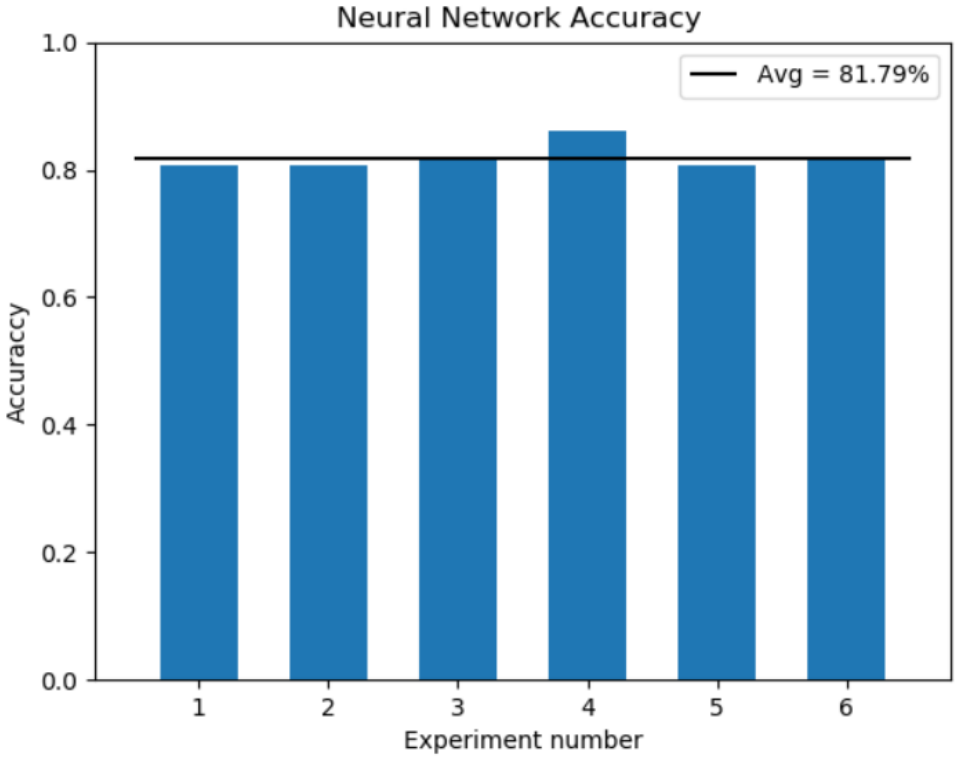


Рис. 39. Диаграмма точности работы нейронной сети.

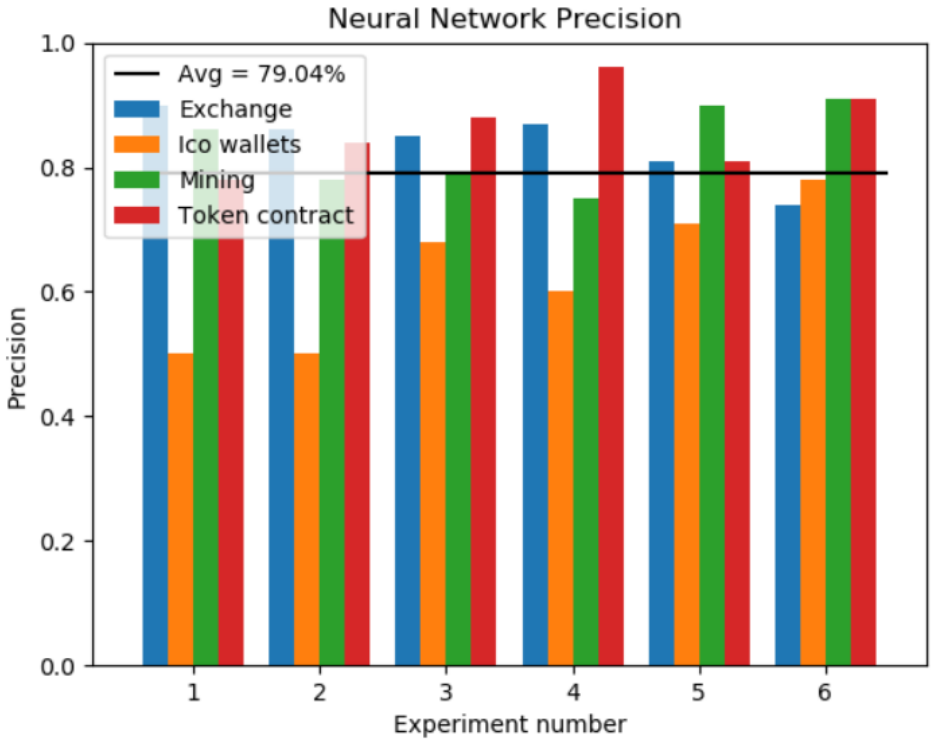


Рис. 40. Диаграмма значений precision нейронной сети.

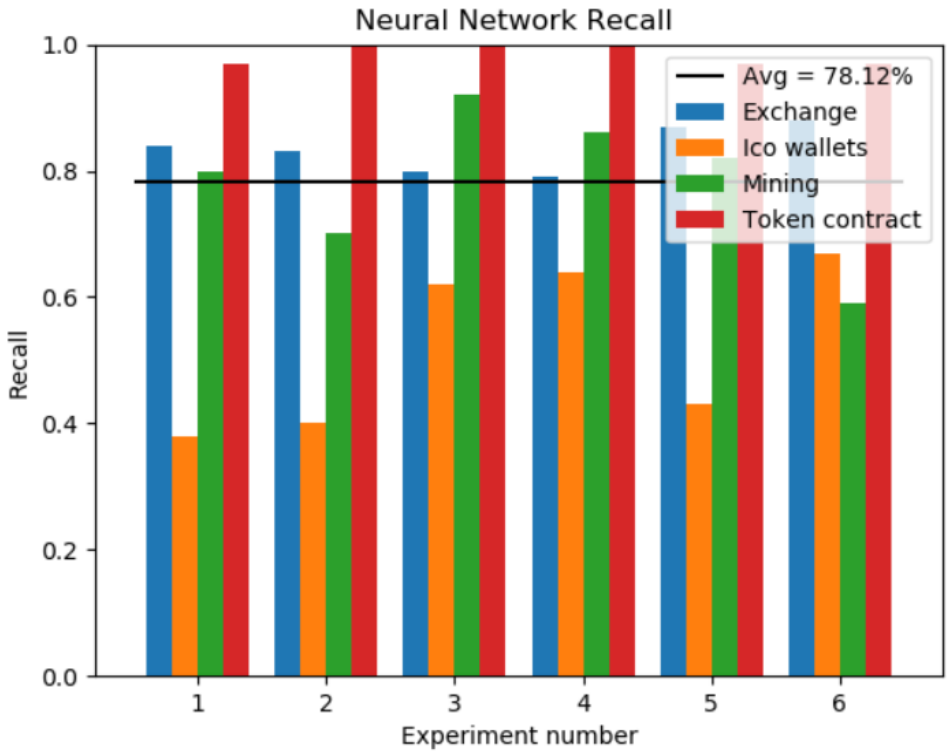


Рис. 41. Диаграмма значений recall нейронной сети.

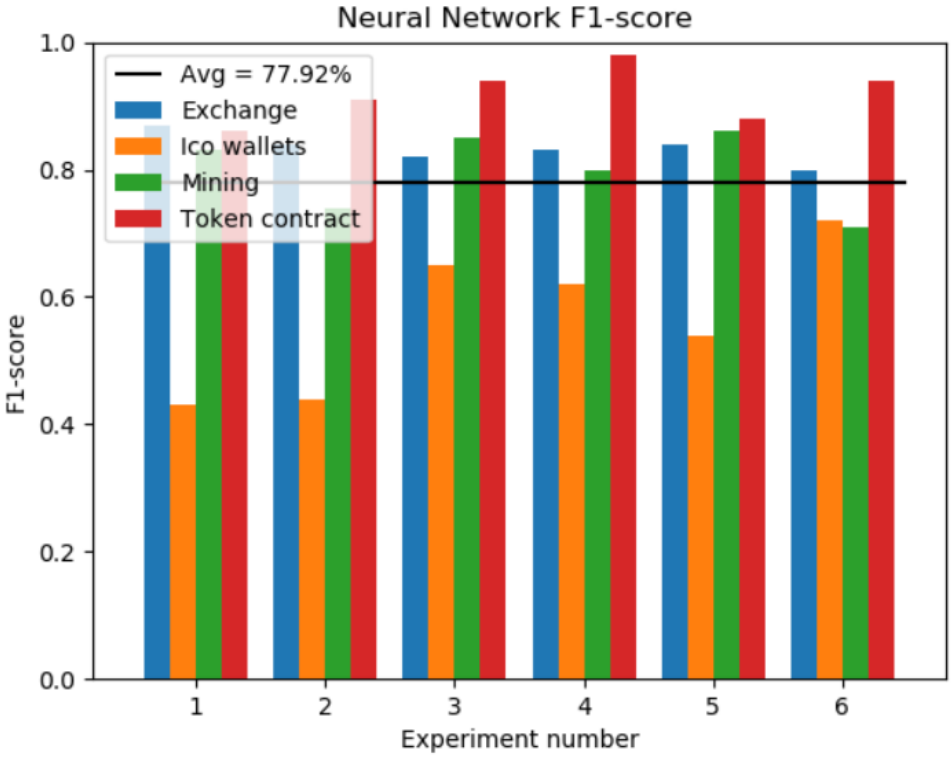


Рис. 42. Диаграмма значений F1-меры для нейронной сети.

Выводы

В данной работе был рассмотрен вопрос классификации адресов блокчейна Ethereum по их поведению в сети. Из списков транзакций адресов были выделены различные метрики и собран датасет для обучения классификаторов.

Задача классификации была решена различными методами:

- Метод k-ближайших соседей,
- Метод опорных векторов,
- Решающие деревья,
- Случайный лес,
- Градиентный бустинг,
- Искусственные нейронные сети.

Доступ к исходному коду классификаторов и программе для генерации датасета можно получить по ссылке: https://github.com/drsanches/masters-work/.

Анализ результатов работы методов показал, что наилучшим образом с данной задачей справляются случайные леса и градиентный бустинг. Также нейронные сети показали хорошие результаты работы на всех классах, за исключением класса “ICO Wallets”. Таким образом, не имеет смысла использовать для решения поставленной задачи такой мощный инструмент, как нейронные сети. Однако, в дальнейшем можно рассмотреть гибридную систему классификации, основанную на случайных лесах и градиентном бустинге с применением нейронных сетей для более точной классификации объектов “Exchange”, “Mining” и “Token contract”.

Также точность работы классификаторов может быть увеличена путем расширения датасета. Архитектура нейронной сети проектировалась оптимальным образом, было проведено множество экспериментов, однако, не

исключена ее модернизация, что представляет простор для дальнейших исследований темы.

Заключение

В данной работе была поставлена задача классификации адресов сети Ethereum по заранее известным типам. Были рассмотрены различные методы искусственного интеллекта для решения задачи классификации.

В результате работы был собран датасет для обучения моделей и произведена оценка пригодности рассматриваемых алгоритмов для решения поставленной задачи по различным метрикам. Вычислительные эксперименты позволили определить наиболее подходящие для решения задачи методы. Для полученных моделей были определены степени важности признаков, используемых для классификации. Также были предложены направления дальнейшего исследования рассматриваемой области. Учитывая дефицит работ по теме классификации адресов блокчейн-сетей, данная работа может быть полезна для изучения указанной темы.

Список литературы

1. A Gentle Introduction to Blockchain Technology [Электронный ресурс]: URL:https://bitsonblocks.net/2015/09/09/gentle-introduction-blockchain-technology (дата обращения: 12.04.2020).
2. Can hashgraph succeed blockchain as the technology of choice for cryptocurrencies? [Электронный ресурс]: URL:https://www.thehindu.com/sci-tech/technology/can-hashgraph-succeed-blockchain-as-the-technology-of-choice-for-cryptocurrencies/article23348176.ece (дата обращения: 12.04.2020).
3. Kiran Kumar Kondru, R Saranya. Directed Acyclic Graph-based Distributed Ledgers – An Evolutionary Perspective. // International Journal of Engineering and Advanced Technology (IJEAT), ISSN: 2249 – 8958, Volume-9 Issue-1, 2019.
4. Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. [Электронный ресурс]: URL:https://bitcoin.org/bitcoin.pdf (дата обращения: 3.03.2020).
5. Ethereum is a global, open-source platform for decentralized applications. [Электронный ресурс] URL:https://ethereum.org (дата обращения: 5.03.2020).
6. Nick Szabo. Smart Contracts: Building Blocks for Digital Markets. [Электронный ресурс] URL:http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html (дата обращения: 7.04.2020).
7. Solidity. Language Documentation. [Электронный ресурс] URL:https://solidity.readthedocs.io/en/latest (дата обращения: 7.04.2020).
8. Justin D. Harris, Bo Waggoner. Decentralized & Collaborative AI on Blockchain. // IEEE International Conference on Blockchain, 2019, 368 – 375.

9. Besir Kurtulmus, Kenny Daniel. Trustless Machine Learning Contracts; Evaluating and Exchanging Machine Learning Models on the Ethereum Blockchain [Электронный ресурс]. URL:https://algorithmia.com/research/ml-models-on-blockchain (дата обращения: 14.05.2020).
10. Ajay Singh. Anomaly detection in the Ethereum network // A thesis for the degree of Master of Technology / Indian Institute of Technology Kanpur, 2019.
11. Chen F., Wan H., Cai H., Cheng G. Machine Learning in/for Blockchain: Future and Challenges // arXiv preprint arXiv:1909.06189, 2019.
12. McMahan H. B., Moore E., Ramage D., Hampson S. Communication-Efficient Learning of Deep Networks from Decentralized Data // arXiv preprint arXiv:1602.05629, 2016.
13. Matthias De Aliaga. Classifying Ethereum users using blockchain data. [Электронный ресурс] URL:https://medium.com/tokenanalyst/classifying-ethereum-users-using-blockchain-data-dd6edb867de3 (дата обращения: 8.04.2020).
14. Will Price. Clustering Ethereum Addresses. [Электронный ресурс] URL:https://towardsdatascience.com/clustering-ethereum-addresses-18aeca61919d (дата обращения: 8.04.2020).
15. Scikit-learn. Choosing the right estimator. [Электронный ресурс] URL:https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html (дата обращения: 26.04.2020).
16. Machine Learning Algorithm Cheat Sheet for Azure Machine Learning designer. [Электронный ресурс] URL:https://docs.microsoft.com/en-us/azure/machine-learning/algorithm-cheat-sheet (дата обращения: 26.04.2020).

17. Which machine learning algorithm should I use? [Электронный ресурс] URL:https://blogs.sas.com/content/subconsciousmusings/2017/04/12/machine-learning-algorithm-use (дата обращения: 26.04.2020).
18. Introduction to k-Nearest Neighbors: A powerful Machine Learning Algorithm (with implementation in Python & R). [Электронный ресурс] URL:https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering (дата обращения: 3.05.2020).
19. Introduction to Support Vector Machines. [Электронный ресурс] URL:https://medium.com/@LSchultebraucks/introduction-to-support-vector-machines-9f8161ae2fcb (дата обращения: 3.05.2020).
20. How decision tree algorithm works. [Электронный ресурс] URL:https://dataaspirant.com/2017/01/30/how-decision-tree-algorithm-works (дата обращения: 3.05.2020).
21. Quinlan, J. R. Induction of Decision Trees. // Machine Learning 1: 81-106, 1986.
22. Quinlan, J. R. C4.5: Programs for Machine Learning // Morgan Kaufmann Publishers, 1993.
23. Ho T. K. Random Decision Forests. // Proceedings of 3rd international conference on document analysis and recognition, vol. 1, 278 – 282, IEEE, 1995.
24. Friedman J. H. Greedy Function Approximation: A Gradient Boosting Machine. // The Annals of Statistics 2001, Vol. 29, No. 5, 1189 – 1232.
25. Everything You Need to Know About Artificial Neural Networks. [Электронный ресурс] URL:https://medium.com/technology-invention-and-more/everything-you-need-to-know-about-artificial-neural-networks-57fac18245a1 (дата обращения: 3.05.2020).

26. McCulloch-Pitts Neuron — Mankind's First Mathematical Model Of A Biological Neuron. [Электронный ресурс] URL:https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1 (дата обращения: 14.04.2020).
27. Rosenblatt F. The Perceptron — a perceiving and recognizing automaton. Report 85-460-1. Cornell Aeronautical Laboratory, 1957.
28. Rumelhart D. E., Hinton G. E., Williams R. J. Learning Internal Representations by Error Propagation // Parallel Distributed Processing, vol. 1, pp. 318—362 / Cambridge, MA, MIT Press. 1986.
29. Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. R., Improving neural networks by preventing co-adaptation of feature detectors // arXiv preprint arXiv:1207.0580, 2012.
30. Ioffe S., Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift // arXiv preprint arXiv:1502.03167, 2015.
31. Tharwat A. Classification assessment methods. // Applied Computing and Informatics, 2018.
32. Etherscan. Label Word Cloud [Электронный ресурс] URL:https://etherscan.io/labelcloud (дата обращения: 2.05.2020).
33. The Selenium Browser Automation Project [Электронный ресурс] URL:https://www.selenium.dev/documentation/en (дата обращения: 20.04.2020).
34. NumPy, SciPy, and Pandas: Correlation With Python [Электронный ресурс] URL:https://realpython.com/numpy-scipy-pandas-correlation-python (дата обращения: 20.04.2020).
35. Scikit-learn. Machine Learning in Python [Электронный ресурс] URL:https://scikit-learn.org/stable (дата обращения: 22.05.2020).

36. Keras: The Python Deep Learning library [Электронный ресурс] URL:https://keras.io (дата обращения: 22.05.2020).
37. TensorFlow. An end-to-end open source machine learning platform. [Электронный ресурс] URL:https://www.tensorflow.org (дата обращения: 22.05.2020).
38. TensorBoard: TensorFlow's visualization toolkit. [Электронный ресурс] URL:https://www.tensorflow.org/tensorboard (дата обращения: 22.05.2020).
39. Scikit-learn. Support Vector Machines. Kernel functions. [Электронный ресурс] URL:https://scikit-learn.org/stable/modules/svm.html#svm-kernels (дата обращения: 22.05.2020).
40. Scikit-learn. MultinomialDeviance. [Электронный ресурс] URL:https://github.com/scikit-learn/scikit-learn/blob/fd237278e895b42abe8d8d09105cbb82dc2cbba7/sklearn/ensemble/_gb_losses.py#L670 (дата обращения: 24.05.2020).
41. Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names. [Электронный ресурс] URL:https://gombru.github.io/2018/05/23/cross_entropy_loss (дата обращения: 24.05.2020).